(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

AD-A240 430

DTIC
ELECTE
SEP 16 1991
S D
D

THESIS

VHDL Simulation of the Implementation
of a Costfunction Circuit

by

Ming Imvidhaya

September, 1990

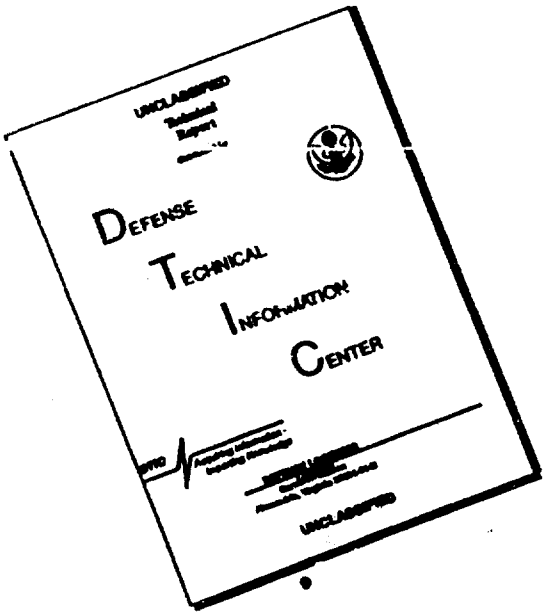
Thesis Advisor: Chin-Hwa Lee

Approved for public release: distribution is unlimited.

91-10679

91 9 16 003

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

REPORT DOCUMENTATION PAGE

1a Report Security Classification: Unclassified	1b Restrictive Markings
2a Security Classification Authority	3 Distribution Availability of Report: Approved for public release; distribution is unlimited.
2b Declassification/Downgrading Schedule	
4 Performing Organization Report Number(s)	5 Monitoring Organization Report Number(s)
6a Name of Performing Organization: Naval Postgraduate School; 6b Office Symbol (if applicable): 62	7a Name of Monitoring Organization: Naval Postgraduate School
6c Address (city, state, and ZIP code): Monterey, CA 93943-5000	7b Address (city, state, and ZIP code): Monterey, CA 93943-5000
8a Name of Funding/Sponsoring Organization; 8b Office Symbol (if applicable)	9 Procurement Instrument Identification Number
8c Address (city, state, and ZIP code)	10 Source of Funding Numbers: Program Element No; Project No; Task No; Work Unit Accession No

11 Title (include security classification): VHDL SIMULATION OF THE IMPLEMENTATION OF A COSTFUNCTION CIRCUIT

12 Personal Author(s): Ming, Imvidhaya

13a Type of Report	13b Time Covered	14 Date of Report (year, month, day)	15 Page Count
Master's Thesis	From To	September 1990	85

16 Supplementary Notation: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms (continue on reverse if necessary and identify by block number)
Field	Group	Subgroup	VHDL, Costfunction, Hardware Description Language

19 Abstract (continue on reverse if necessary and identify by block number)

Since VHDL is a DoD standard hardware description language, it is widely used in the design of logic circuits at different levels. VHDL can be used to do behavioral modeling which is desirable in top-down system design. A costfunction calculation in a graph partition algorithm is used here as an example to test the VHDL design methodology. Subroutines or statements in the software can be implemented into hardware if the subroutines or the statements in that software are suitably grouped. While the design of hardware is considered, high density integration of circuit is also the primary goal. Parts of an old design were condensed using programmable EPLDs which were programmed by commercial software development tools. The methodology of implementation goes from a register transfer language description to data flow design and control flow design. The costfunction calculation was successfully put into 4 EP1800 chips and the design was simulated in VHDL. The primary goal of integration was achieved at the expense of speed. To support the total simulation several behavior models were created. Results of simulation revealed that the adder circuit in the EP1800 can be further improved. Experiences of using VHDL are discussed in this thesis.

20 Distribution/Availability of Abstract	21 Abstract Security Classification
☒ unclassified/unlimited ☐ same as report ☐ DTIC users	Unclassified

22a Name of Responsible Individual	22b Telephone (include Area code)	22c Office Symbol
Chin-Hwa Lee	(408) 646-2190	62Le

DD FORM 1473,84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

Approved for public release; distribution is unlimited.

VHDL Simulation of the Implementation
of a Costfunction Circuit

by

Ming Imvidhaya
Lieutenant Commander, Royal Thai Navy
B.S., Royal Thai Naval Academy 1980

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1990

Author: Ming Imvidhaya
Ming Imvidhaya

Approved by:
Chin-Hwa Lee, Thesis Advisor

Jon T. Butler, Second Reader

Michael A. Morgan, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

Since VHDL is a DoD standard hardware description language, it is widely used in the design of logic circuits at different levels. VHDL can be used to do benavioral modeling which is desirable in top-down system design. A costfunction calculation in a graph partition algorithm is used here as an example to test the VHDL design methodology. Subroutines or statements in the software can be implemented into hardware if the subroutines or the statements in that software are suitably grouped. While the design of hardware is considered, high density integration of circuits is also the primary goal. Parts of an old design were condensed using programmable EPLDs which were programmed by commercial software development tools. The methodology of implementation goes from a register transfer language description to data flow design and control flow design. The costfunction calculation was successfully put into 4 EP1800 chips and the design was simulated in VHDL. The primary goal of integration was achieved at the expense of speed. To support the total simulation several behavior models were created. Results of the simulation revealed that the adder circuit in the EP1800 can be further improved. Experiences of using VHDL are discussed in this thesis.

Accesion For
NTIS CRA&I
DTIC TAB
Unannounced
Justification
By
Distribution /
Availability
Dist | Special
A-1

TABLE OF CONTENTS

I. INTRODUCTION

A. INTRODUCTION

There is strong interest of being able to detect signal tracks in a lofargram in a noisy environment. A lofargram is a two dimensional display of power spectrum with respect to the time axis and the frequency axis of acoustic sources. A marine vessel with man-made noise will show up as tracks in lofargram. One method of detection in a noisy lofargram is to transform this signal detection problem to a graph partition problem. Each pixel of the lofargram corresponds to the node of a graph. The horizontal chaining of the pixel along the time frame corresponds to the edges of the graph. The constraints of track positions are correlated to the graph precedence associated with the edges. The problem of finding maximum signal-to-noise ratio tracks in a lofargram becomes a problem of partitioning the graph which can result in minimum cost [Ref. 1]. The costfunction is defined as:

$$Cf_{i,j}=\gamma\sigma_{i,j}-\eta_i$$

where η_i is the averaged signal power along track i.

$\sigma_{i,j}$ is the averaged noise between track i and track j.

γ is a threshold constant that determines the false alarm rate.

$Cf_{i,j}$ is the incremental cost of partitioning the graph with the new track j given that the last partition of track j is already accomplished. The original objective of achieving maximal signal-to-noise ratio is changed to achieving minimum cost of $Cf_{i,j}$. When track i is located at the right cut of the graph, the cost of $Cf_{i,j}$ will be minimized. The graph partitioning algorithm by Jensen [Ref. 2] enumerates all possible partitions of a graph and evaluates them efficiently using a derived tree and the costfunction. All possible partitions of a graph are enumerated in the structure of a tree which is generated sequentially using an algorithm described in detail by Jensen. Moreover, the solution of the partitioning algorithm is optimal with respect to the costfunction used. The problem of partitioning was formulated as a dynamic programming problem which is an equivalent search of the tree. The tree generated by this algorithm contains all the information required to locate the optimum partition of the graph. This is a dynamic programming approach to find a global optimum solution.

An additive step costfunction for a set of nodes is defined above so that the total cost is a sum of the individual step costs. In effect, all possible pairs of partitions are evaluated using an additive costfunction. The aim is to determine the minimal cost over the space of partitions of the graph. At the end, the total cost of a

solution is given by the sum of the cost of all the individual partitions.

The calculation of the step costfunction occurs frequently in the algorithm and this calculation consumes a major portion of the execution time. Wu, et al, of the Naval Research Laboratory have designed a hardware board to calculate the step costfunction. This hardware design is called "PARTITION".

The contribution of this research is the development of a means to use the VHDL hardware description language to model and simulate the functions of the existing circuit. The circuit to be studied performs the mathematical calculation of the COSTFUNCTION which is a subroutine in the graph partitioning algorithm. The COSTFUNCTION calculation can be performed in either a C language program or an implemented logic circuit. The COSTFUNCTION circuit was separated from a bigger graph partitioning algorithm written in C and implemented in TTL, multiplier chip, and PAL chips. The hardware of the graph partition algorithm is called the PARTITION circuit. Parts of the circuit will be implemented in condensed form using several Erasable Programmable Logic Devices (EPLD). In particular, the high density (2000 gates) EPLD; i.e. EP1800 is used for this purpose.

B. VHDL HARDWARE DESCRIPTION LANGUAGE

VHDL stands for VHSIC Hardware Description Language. It is a new hardware description language developed and standardized

by the U.S. Department of Defense for documentation and specification of CAD microelectronics design [Ref. 3]. The language was developed to address a number of recurrent problems in the design cycles, exchange of design information and documentation of digital hardware. VHDL is technology independent and is not tied to a particular simulator or logic value set. Also it does not force a design methodology on a designer [Ref. 4]. Many existing hardware description languages can operate at the logic and gate level. Consequently, they are a low-level logic design simulators. While VHDL is perfectly suited to this level of description, it extends beyond this to higher behavioral levels.

The study here using VHDL to describe the costfunction logic circuit revealed both the advantages and the disadvantages of the simulated circuit as well as the limitations of the procedure of density integration using EP1800.

C. ALTERA ERASABLE PROGRAMMABLE LOGIC DEVICES

ALTERA development tools are available to program the Erasable Programmable Logic Device (EPLD) [Ref. 5]. An EPLD is a combination of CMOS devices and EPROM devices. The family of EPLDs spans the range of density from 300 to over 2000 gates. The ALTERA CAD tool, A+PLUS, is used in this research. As shown in Figure 1, the package allows mixed format design entries; Boolean equations, state machine,

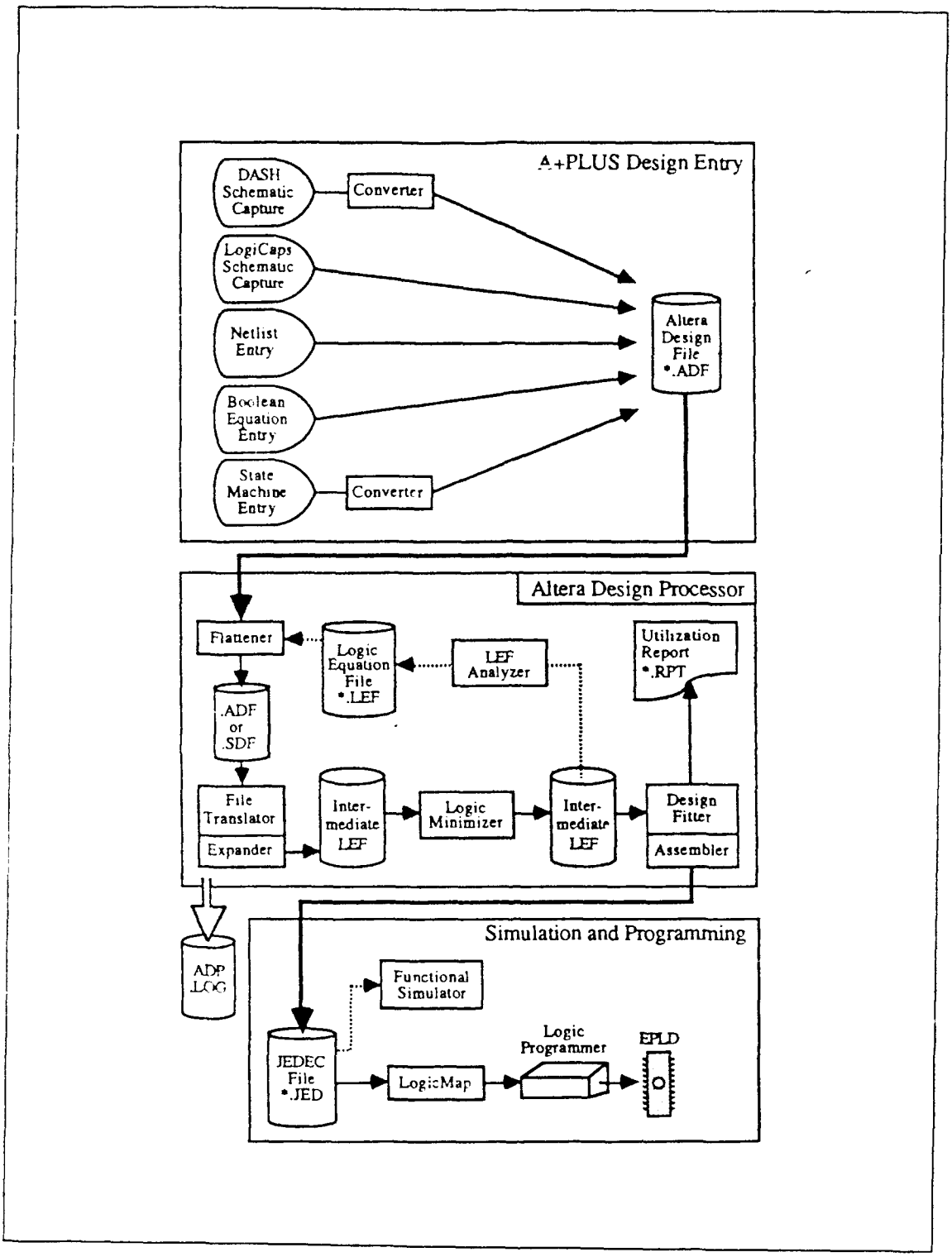


Figure 1. A+PLUS Block Diagram
(adapted from ALTERA data book)

netlist, and schematic capture. "Design Processing" performs logic minimization, automatic device fitting, utilization reporting, and creation of standard JEDEC programming files. Device fitting is the PLD equivalent to an automatic place and route capability. The ALTERA EPLD development tools are installed on an IBM AT computer for this research.

By using the ALTERA development tools, part of the PARTITION schematics design are translated into JEDEC files. These JEDEC files will be read into the VHDL structural model of the EP1800 to perform the programmed EPLD function.

D. COSTFUNCTION

Sonar lofargrams tend to be noisy and of low contrast. The feature of interests is a line on a track. The track detection problem is one of translating the image processing problem into a graph such that the nodes of the graph correspond to the pixels of the lofargram. Any cut through the graph generates a track at a length as long as the number of time lines in the graph.

The graph weights are derived from some pre-defined measure such as display pixel intensity value, which is an analogue of signal strength. Target tracks are manifested as cuts through this graph. The optimal partitioning of the graph can be based upon some objective criterion such as the signal-to-noise ratio.

The costfunction is based roughly on signal-to-noise ratio:

$$Cf_{i,j}=\gamma\sigma_{i,j}-\eta_i$$

This is a weighted noise estimate less the signal estimate. The track t_i is modelled as being one pixel wide. The signal estimate η is obtained by integrating the graph weights along the track path t_j. The noise estimate $\sigma_{i,j}$ is determined from the mean weight of the nodes between t_i and t_j. The scaling constant γ can be varied to change the detection threshold.

The costfunction formula above can be mapped directly as

$$\text{costfunction}(i,j)=\frac{\text{table}[i]-\text{table}[j]}{\text{size}[i]-\text{size}[j]}-\text{acc}[i]$$

if $\gamma = 1$. The array table[] stores the accumulated pixel intensity to the current track position and array size[] stores the accumulated number of nodes to the current track. The signal estimate for each track path is stored in array acc[]. This formula was written in a C language program as follows:

```
int costfunction(i,j)
VERTEX i,j;
{
  register int icost;
     icost =  (int) (table[i] - table[j]);
     icost /= (int) (size[i] - size[j]);
     return (icost - (int) acc[i]);
}
```

In order to obtain the faster speed operation of this subroutine, the above software is implemented in hardware. A

circuit of the costfunction, as shown in Figure 2, consists of TTL ICs, PROMs, RAMs and PAL devices. Most PAL devices produce control signals to manipulate the operations of the circuit in coordinated complex timing sequences.

1. RTL description of the operations on PCB

The Register Transfer Language (RTL) description of the costfunction circuit consists of 10 groups of register transfers as shown in Figure 3. A group represented by vertically linked indicates the concurrent transfer. When a transfer is made into memory, the Memory Address Register (MAR) is used. The transfer from the location in memory identifed by the memory address register is specified with square brackets. Each buffer register is associated with a name REG. These registers usually consist of the D type flip-flops.

2. Operations of the Costfunction Circuit

Initially, when a main program is executed (in IBM PC or compatible) the computer communicates directly to the costfunction Printed Circuit Board (PCB) in order to load values of array ACC[i] and array TABLE[i] into ACC RAM (IC37,35) and TABLE RAM (IC36,34) respectively in Figure 2. The loading is performed in IC40 by decoding the PC address bus into control signals. These control signals are interpreted by IC08 to generate asynchronous control sequences. After the control signals are established, the data

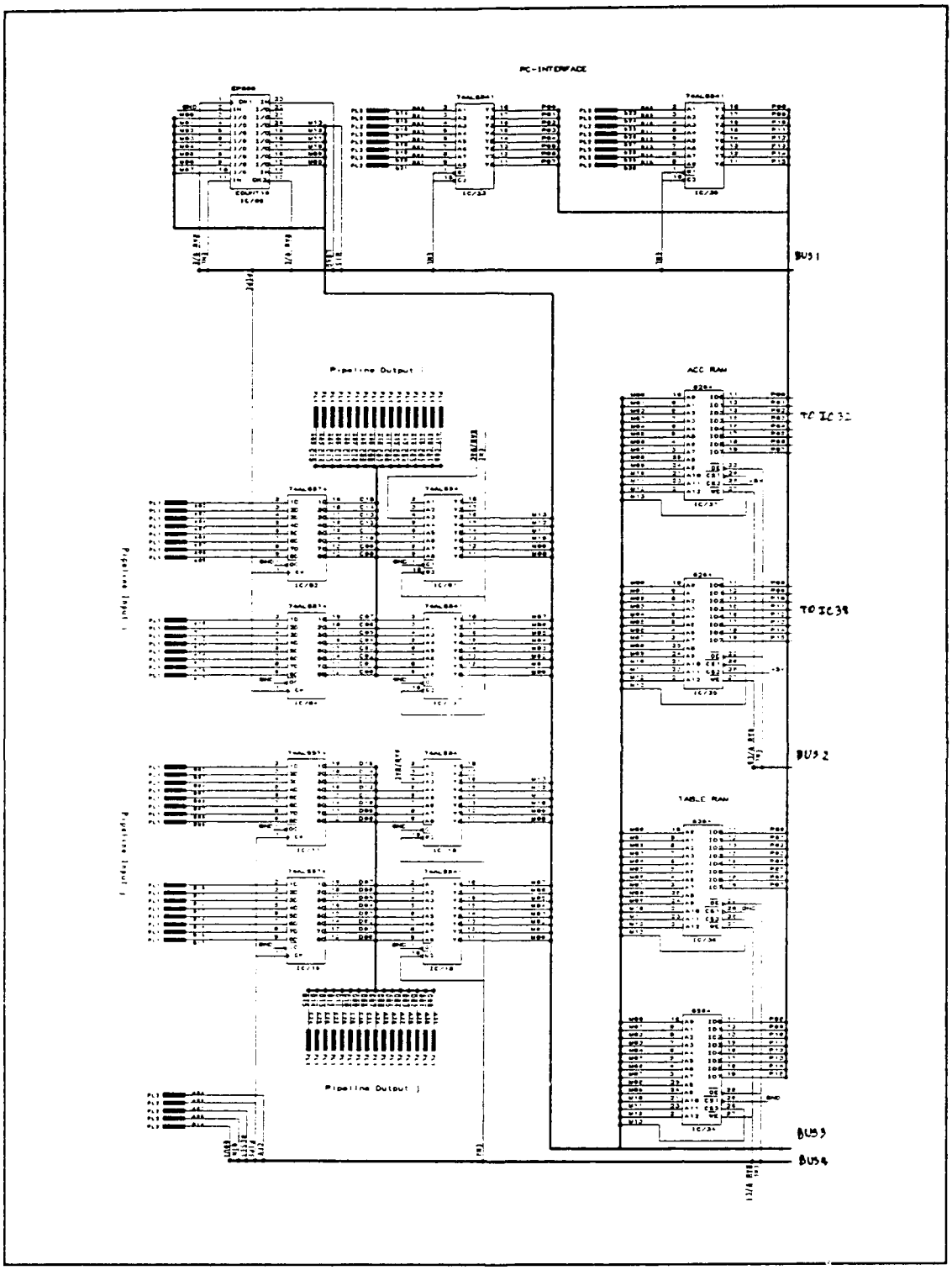


Figure 2. Costfunction Circuit

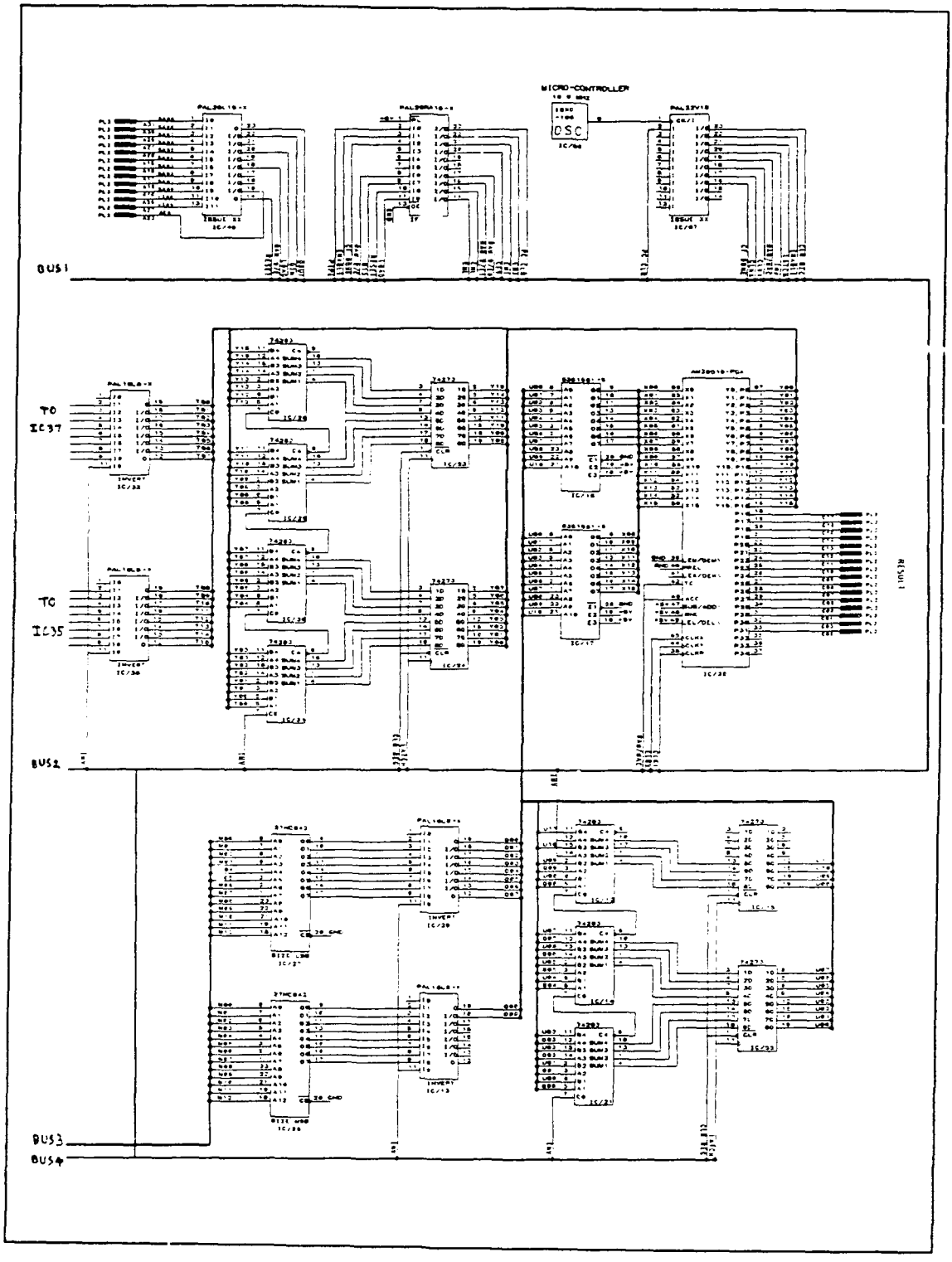


Figure 2(con't). Costfunction Circuit

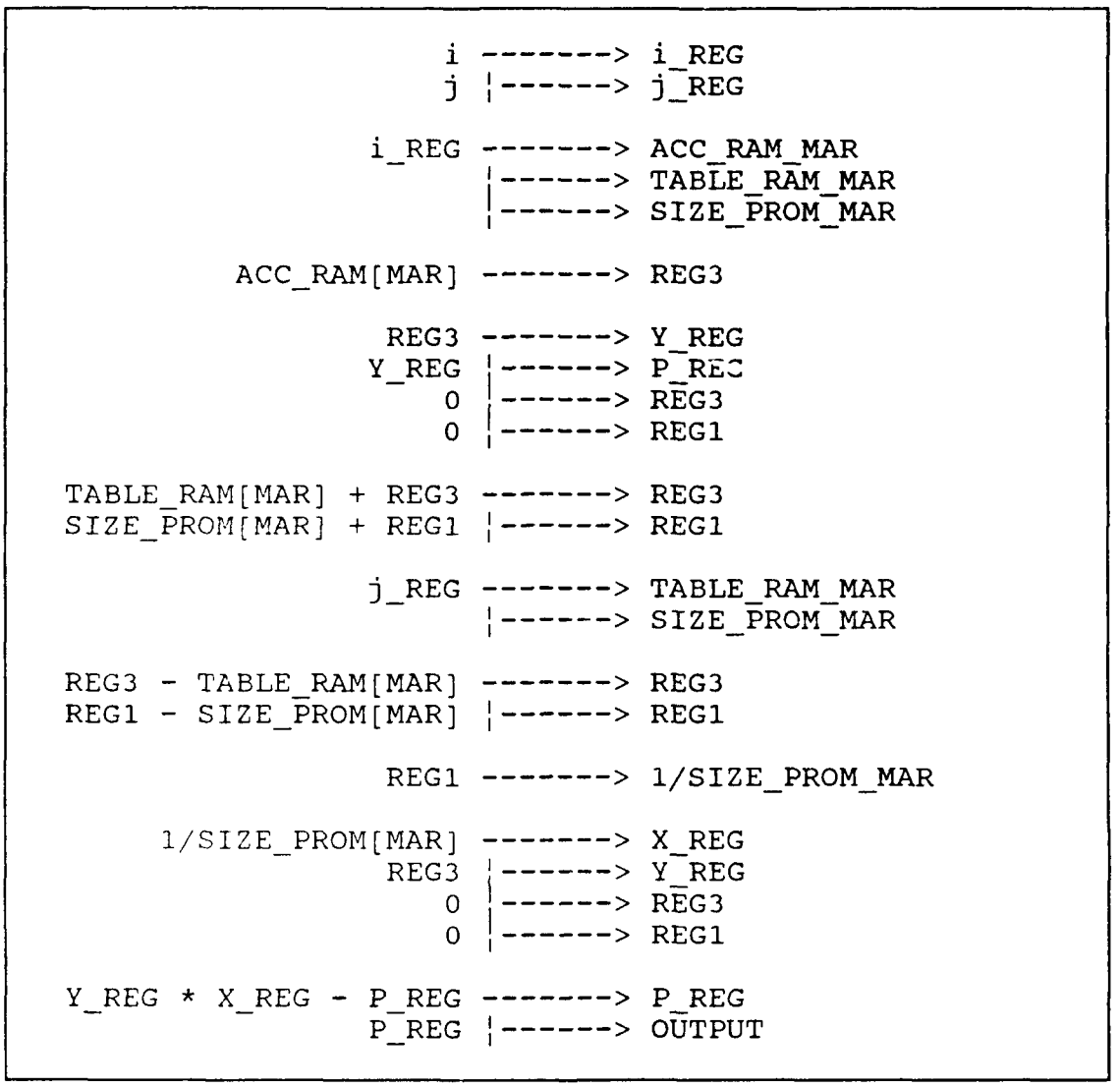


Figure 3. RTL of the original Costfunction circuit

is transferred from the PC data bus into the ACC RAM and the TABLE RAM, respectively, via IC33 and IC39. After loading is completed, the costfunction PCB is ready to perform the costfunction subroutine as discussed above.

The basic operations of the costfunction circuit can be described in RTL as shown in Figure 3.

When the subroutine is called, value i and j are passed simultaneously through registers IC02,04 and IC11,19 and latched into IC01,03 and IC10,18 respectively. This is shown as the first RTL group in Figure 3. Then the value of i is latched through IC01,03 in order to go to the ACC RAM (IC35,37), the TABLE RAM (IC36,34), and the SIZE PROM (IC26,27) as an address shown in the second group of RTL in Figure 3. The ACC RAM and the SIZE PROM are enabled first. After ACC[i] is accessed and the value is passed from RAM via adder circuit into the Y register and the accumulator of the AM29510, the TABLE RAM is enabled to access the value at TABLE[i] location. The outputs from memory devices are passed through IC32,38 and IC20,13 without negation into the adder circuits. After the registers of the adder is latched, the output of the adder registers are TABLE[i] and SIZE[i]. This is described in the fifth group of RTL in Figure 3.

Next, the value of j is passed in the same way as i to access TABLE[j] and SIZE[j] as shown in the sixth group and the seventh group of the RTL description in Figure 3. Then the outputs from both memories are negated by the inverter PAL (IC32,38 and IC20,13) and passed into the adder circuits. This time each adder performs two's complement addition and the results after the latching are the value of (TABLE[i]-TABLE[j]) and (SIZE[i]-SIZE[j]) in IC23,24 and IC15,22.

The value of (SIZE[i]-SIZE[j]) is used to access the PROM (IC16,17) to get the inverted value. Then the output from

the PROM is gated into the X register of the multiplier (IC25) by signal CLKX. At the same time, the value of (TABLE[i]-TABLE[j]) is also gated into the Y register by signal CLKY. This is shown as the ninth group of the RTL description in Figure 3. The result of the multiplication is subtracted from the previous value in the accumulator (ACC[i]), and the result is stored back into the accumulator.

After the CLKP signal, the result of the costfunction is available from AM29510 at the P register together with the CF_DONE signal and CF_V signal from IC07 and IC08, respectively.

E. OVERVIEW OF THE THESIS

This thesis is divided into five chapters. Chapter I gives the introduction of the costfunction circuit, VHDL, and ALTERA software package. Discussion of how the partition of the costfunction circuit was made is discussed in Chapter II. Chapter III includes VHDL behavior modeling of some specific ICs in the circuit. Simulation of the total costfunction circuit and results are discussed in Chapter IV and the improvement of the design is also included. Finally, Chapter V gives the conclusions and the suggestions of possible future research.

II. FUNCTIONS OF THE PARTITION CIRCUIT DESIGN

A. C PROGRAM TO RTL AND SCHEMATIC DIAGRAM

The costfunction calculates the signal to noise ratio of each track partition between t_i and t_j and gives the result back to the graph partition algorithm. The costfunction subroutine is written in C language. This subroutine is called from the program PARTITION which performs the graph partition algorithm. The program is shown in Figure 4.

In order to implement this subroutine in hardware, each statement in the program must be considered carefully. Normally, the internal hardware activities are consisted of both the control flow and the data flow. The control flow can be implemented easily if the data flow is well described in the RTL. The RTL shows sequences of operations of each functional modules at different time [Ref. 6]. A simple block diagram of the costfunction hardware is shown in Figure 5. This block diagram is more abstract than the original circuit shown in Figure 2. It shows how the EPLDs are used in the implementation. Data paths are implemented according to this partition. The modules of the block diagram, their names, and their function are as follows:

- CHIP4i: This is an input buffer for value i.
- CHIP4j: This is an input buffer for value j.

```
#include "part.h"
#include <stdio.h>

void partition( nodecount )
int nodecount;
{
        register int    i, j, c;
        BOOLEAN         status;
        int             cost[MAXNODE];

    FILE            *fp, *fopen();
    int             minval;

        int             costfunction();

        for (i=0;i<nodecount;i++)
        {
                cost[i] = BIGINT;
                opt[i]=nullset;
        }

        for (i=1;i<nodecount;i++)
    {
                for( j=0; j<i; j++ )
                {
                        status=TRUE;
                        for(c=0;c<lines;c++)
                if(setsize[i][c]<=setsize1[j][c])
                {
                    status=FALSE;
                    break;
                }

                        if (status)
                        {
                                c = costfunction( i, j ) + cost[j];

                                if( (cost[i]) >= c )
                                {
                                        cost[i] = c;
                                        opt[i] = j;
                                }
                        }
                }
    }
}

int costfunction( i, j )        /* cost of segment */
VERTEX i, j;
{
        register int icost;

        icost = (int) (table[i]-table[j])*6;
        icost /= (int) (size[i]-size[j]);

        return( icost - (int) acc[i] );
}
```

Figure 4. Program PARTIT.C

- ACC RAM memory: This memory is used to store values of the array ACC.

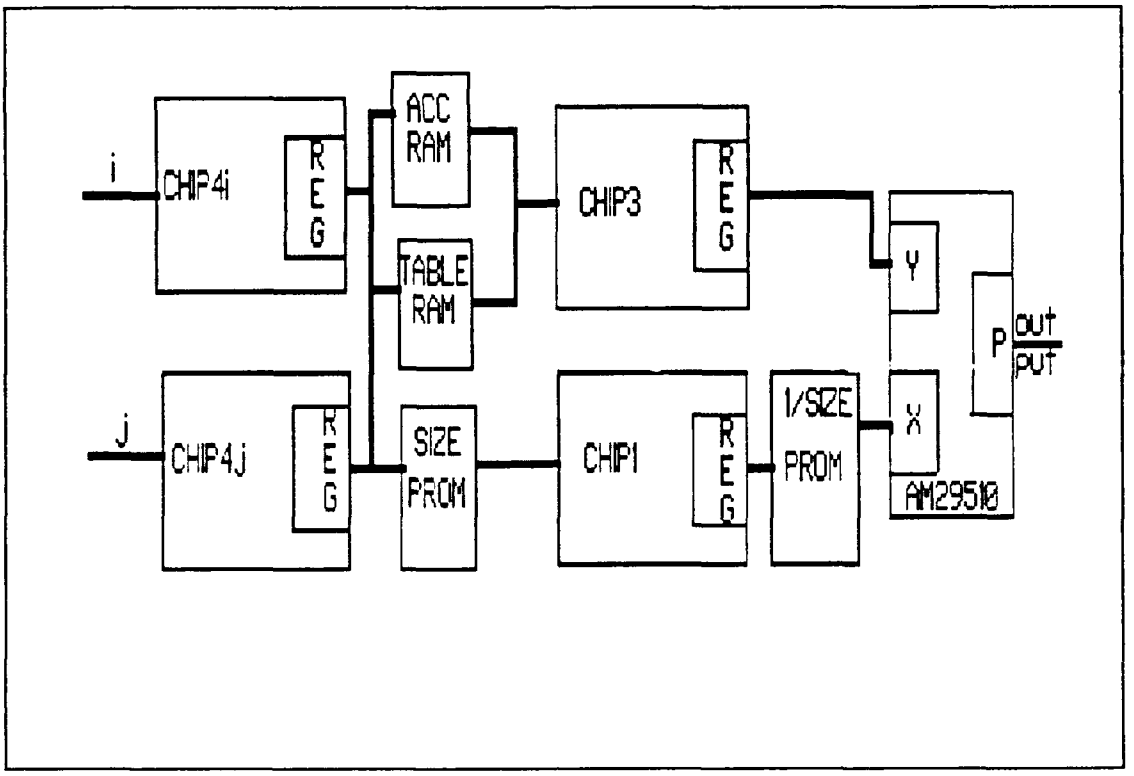


Figure 5. Simple block diagram of the costfunction

- TABLE RAM memory: This memory is used to store values of the array TABLE

- SIZE PROM memory: This memory contains the constant values of array SIZE. The memory chips chosen here are Programmable Read Only Memory (PROM).

- CHIP3: This is an EP1800 device which is programmed to pass the value of ACC[i] and also perform TABLE[i] - TABLE[j] operation. There is an internal output register.

- CHIP1: This is an EP1800 device which is programmed to perform SIZE[i] - SIZE[j] operation. This chip also has an internal output register.

- 1/SIZE PROM memory. This is a PROM memory which contains the inversion of the input value from CHIP1.

- AM29510: This module has the responsibility for doing the multiplication needed in the algorithm. It will multiply,

then add or subtract the value to or form the accumulator. There are three registers internal to the system, two input registers (X,Y) and one output register (P).

Once the data path is decided for the basic algorithm, the RTL can be written as shown in Figure 6. Once the RTL is completed, a state diagram associated with the control flow can be identified. It's obvious that this state diagram can be used to implement an appropriate controller directly.

B. RTL DESCRIPTION FOR THE REDESIGNED COSTFUNCTION PCB

The original costfunction circuit in Figure 2 is previously described in the block diagram of Figure 5. This diagram breaks the circuit into several blocks. The redesigned costfunction circuit will implement some of these blocks in EPLDs.

The RTL in Figure 6 represents the operations of the redesigned costfunction circuit using EPLD. The RTL description identifies the appropriate calculations as well as the possible parallelism of the events. The system waits for the value of i and j to be loaded into the input buffer registers, at this point in time the data processing begins. The value of i from CHIP4i register is loaded as an address into the ACC RAM, the TABLE RAM, and the SIZE PROM concurrently as shown in step two in Figure 6. Then the CHIP3 register is loaded with ACC[i] from the ACC RAM in step three. The next step is to transfer the content of the CHIP3 register

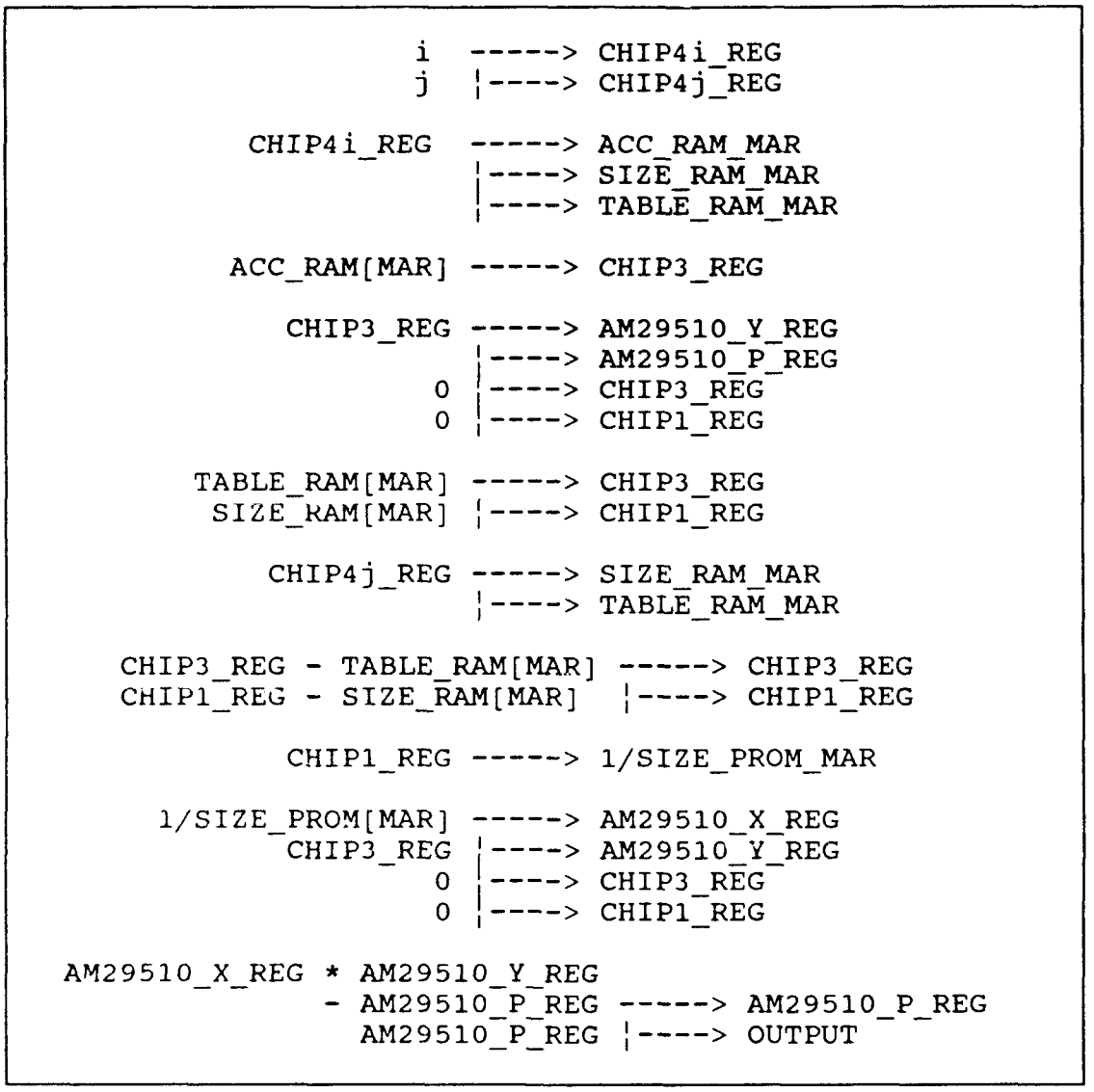


Figure 6. RTL of the EPLD implementation of the costfunction

to the Y register and simultaneously clear the CHIP3 register and the CHIP1 register. The value of Y register is also transferred to the P register in step four. At the next step the CHIP3 register and the CHIP1 register are loaded with TABLE[i] and SIZE[i] from the TABLE RAM and the SIZE PROM respectively. The next group of transfers, step six, provide

the value of j from CHIP4j as an address into the TABLE RAM and the SIZE PROM. Then, the values of TABLE[j] and SIZE[j] are subtracted from the previous TABLE[i] and SIZE[i], and the result are placed into their registers as shown in step seven. Next, in step eight, the value from CHIP1 register is loaded into the 1/SIZE PROM as an address. In step nine, the value from the 1/SIZE PROM is loaded into X register. At the same time, the value from the CHIP3 register is also loaded into the Y register. At this point, the CHIP3 register and the CHIP1 register are also cleared. The final calculation is then done, and the result is transferred to the P register as well as the output port.

C. FROM RTL DESCRIPTION TO STATE DIAGRAM

The state diagram, as shown in Figure 7 represents a control section that can manipulate the control signals in such a way that the simultaneity specified in the RTL is maintained. The RTL description shows the sequence of register transfer operations. In each step of operation, some events can occur concurrently. Therefore, the control signals generated in each step of the state machine can be identified as follows:

- State 0 accepts value of i and j. The ENABLE_H signal is asserted high so that the i is transferred to the ACC RAM, the TABLE RAM, and the SIZE PROM as addresses. The ENABLE_H signal is maintained until the end of state 4.

Present state	Activities
STATE 0	ENABLE_H
STATE 1	ENABLE_H WAIT_H
STATE 2	ENABLE_H LATCH_H CLKX_H
STATE 3	CLEAR_L ENABLE_H RAM/MAC_H CLKY_H
STATE 4	ENABLE_H RAM/MAC_H WAIT_H
STATE 5	LATCH_H RAM/MAC_H
STATE 6	INV_H RAM/MAC_H WAIT_H
STATE 7	LATCH_H INV_H RAM/MAC_H
STATE 8	CLEAR_L RAM/MAC_H CLKX_H CLKY_H
STATE 9	CLEAR_L
STATE 10	CLEAR_L CLKY_H CF_DONE_H

Figure 7. State diagram of costfunction

- State 1 is a wait state which allows a delay for the memory to be accessed. The WAIT_H signal is asserted from the "Buried" state bit inside the EP1800. This bit is used to insert the wait states to satisfy the setup time

requirement between the memory address valid and the accumulator latches.

- State 2 asserts the LATCH signal which causes the ACC[i] to be placed into the CHIP3 register. The control status register of AM29510 are also loaded in this state by asserting the CLK_X signal.

- State 3 causes three things to happen. The CLK_Y signal is asserted to transfer the ACC[i] from the CHIP3 register into the Y register of AM29510. The value goes to the P register which is the accumulator of AM29510 as well. The CLEAR signal is also asserted to clear the registers of CHIP3 and CHIP1. At the same time, the RAM/MAC signal is asserted to enable the accessing of the TABLE RAM. This signal also set the two's complement input mode and accumulator mode of the AM29510. The RAM/MAC_H signal is maintained until the end of state 8.

- State 4 is also a wait state which allows the delay for the TABLE[i] and SIZE[i] in RAM and PROM to be accessed.

- State 5 asserts the LATCH signal to place TABLE[i] and Size[i] into the CHIP3 register and the CHIP1 register. The ENABLE is not asserted which causes the j to be addressed into the TABLE RAM and the SIZE PROM.

- State 6 is also a wait state for memory accessing. In this state the INV signal is asserted high to get the two's complement value of -TABLE[j] and -SIZE[j] respectively.

- State 7 asserts the LATCH signal in order to place the TABLE[i] - TABLE[j] and the SIZE[i] - SIZE[j] into the CHIP3 register and the CHIP1 register. The value in CHIP1 register causes the accessing of the 1/SIZE PROM memory.

- State 8 causes two things to happen. The CLK_X and CLK_Y are asserted to load the values from the CHIP3 register and the 1/SIZE PROM into the X register and the Y register. The CLEAR signal is also asserted to clear the CHIP3 register and the CHIP1 register afterwards.

- State 9 is a wait state for the internal calculation inside the AM29510 multiplier.

- State 10 causes the result to be filled into the output register by asserting the CLK_Y signal. The CF_DONE is also asserted to indicate the valid value is available at the output port.

D. BACKGROUND OF THE EP1800 STRUCTURE

The EP1800 is an erasable, user-configurable LSI device that has 2100 equivalent gates logic [Ref. 7]. Externally, the EP1800 provides 16 dedicated inputs, 4 of which may be used as system clock inputs. There are 48 I/O pins which may be individually configured for input, output, or bidirectional data flow as shown in Figure 8. Internally, the EP1800 architecture consists of a series of macrocells. Logics are implemented within these cells. Each macrocell contains 3 basic elements: a logic array, a selectable register element, and a tri-state I/O buffer.

The EP1800 is partitioned into four identical quadrants. Each quadrant contains 12 macrocells. Input signals into macrocells can come from the EP1800 internal bus. Macrocell outputs may drive the external pins as well as the internal buses.

Sixteen of the 48 macrocells offer increased speed performance through the logic array. These "Enhance Macrocells" can be used for critical combinatorial logic with short delay paths. There are 4 enhanced macrocells for each quadrant.

Another kind of macrocell provides dual functions. These "Global macrocells" allow implementation of buried logic functions and, at the same time, serving as dedicated input pins. The global macrocells have the same timing characteristics as the general macrocells.

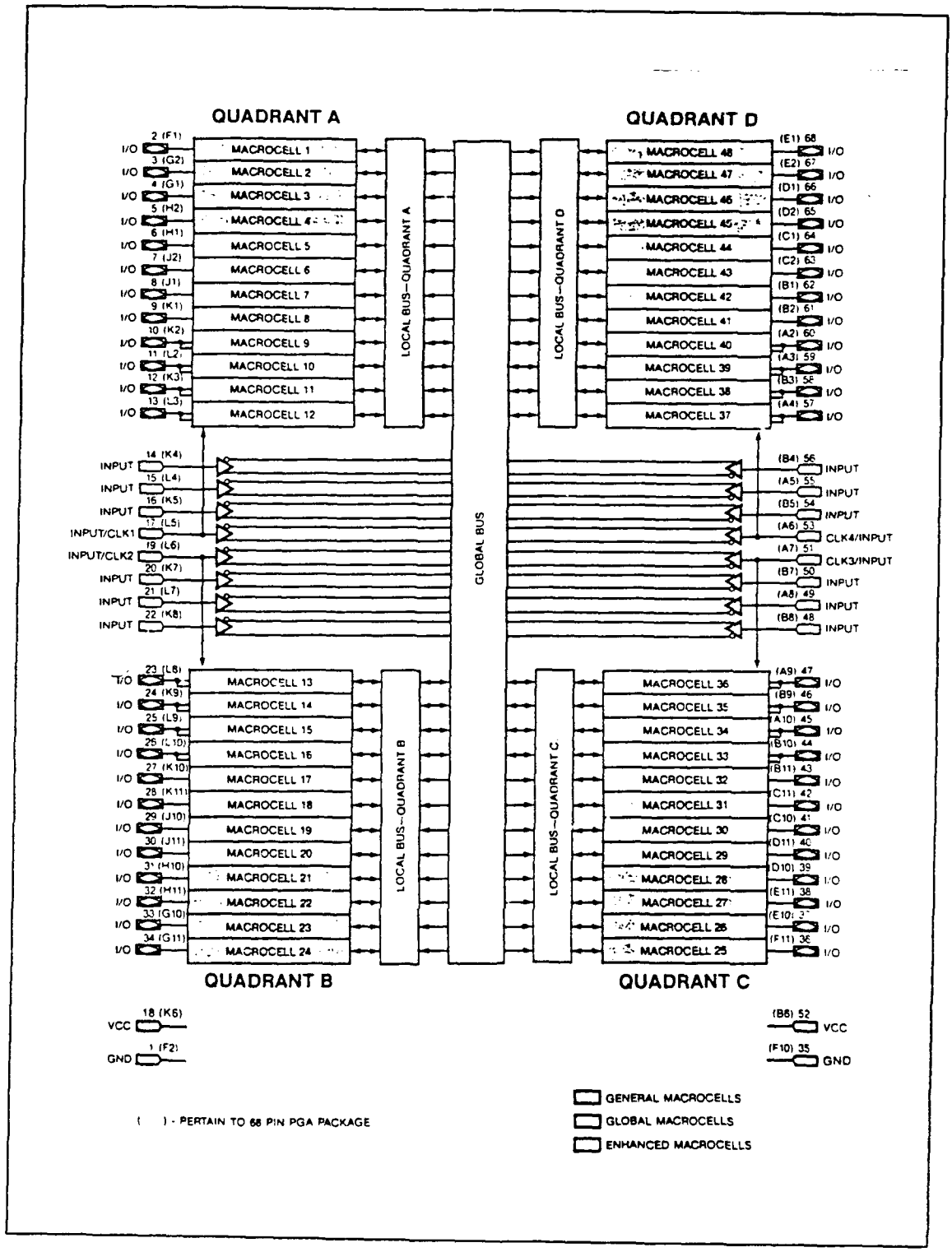


Figure 8. EP1800 block diagram
(adapted from ALTERA data sheet)

A structural model of the EP1800 in VHDL is written by a previous thesis student [Ref. 8]. This model will be used in this research.

E. IMPLEMENTATION OF THE FUNCTION A = A + B

In order to implement this algebraic function in one chip, a full adder circuit and a register must be combined together. Since the current VHDL model of the EP1800 can only represent D type flip-flop registers, the selection of D-FF in the ALTERA design tools is mandatory. A register is necessary to store the value of the previous state (value of A). Then this stored value is fed back to one input of the adder to be added with another input (value of B). The result of the addition can latched into the D-FF register after the rising edge of the clock signal.

By using Schematic Entry in ALTERA software, a 1-bit full adder is built from the ALTERA primitives. ALTERA also has an output primitive which has both a register and a feedback pin. This is called a RORF primitive. The circuit of a 1-bit full adder with RORF is shown in Figure 9. The CLR signal is used to reset the output of the register to zero. As an example, the operation of countup can be done using this adder in the flow chart shown in Figure 10. Initially an input of 1 is set to A. What needs to be done next is to clock the circuit subsequently.

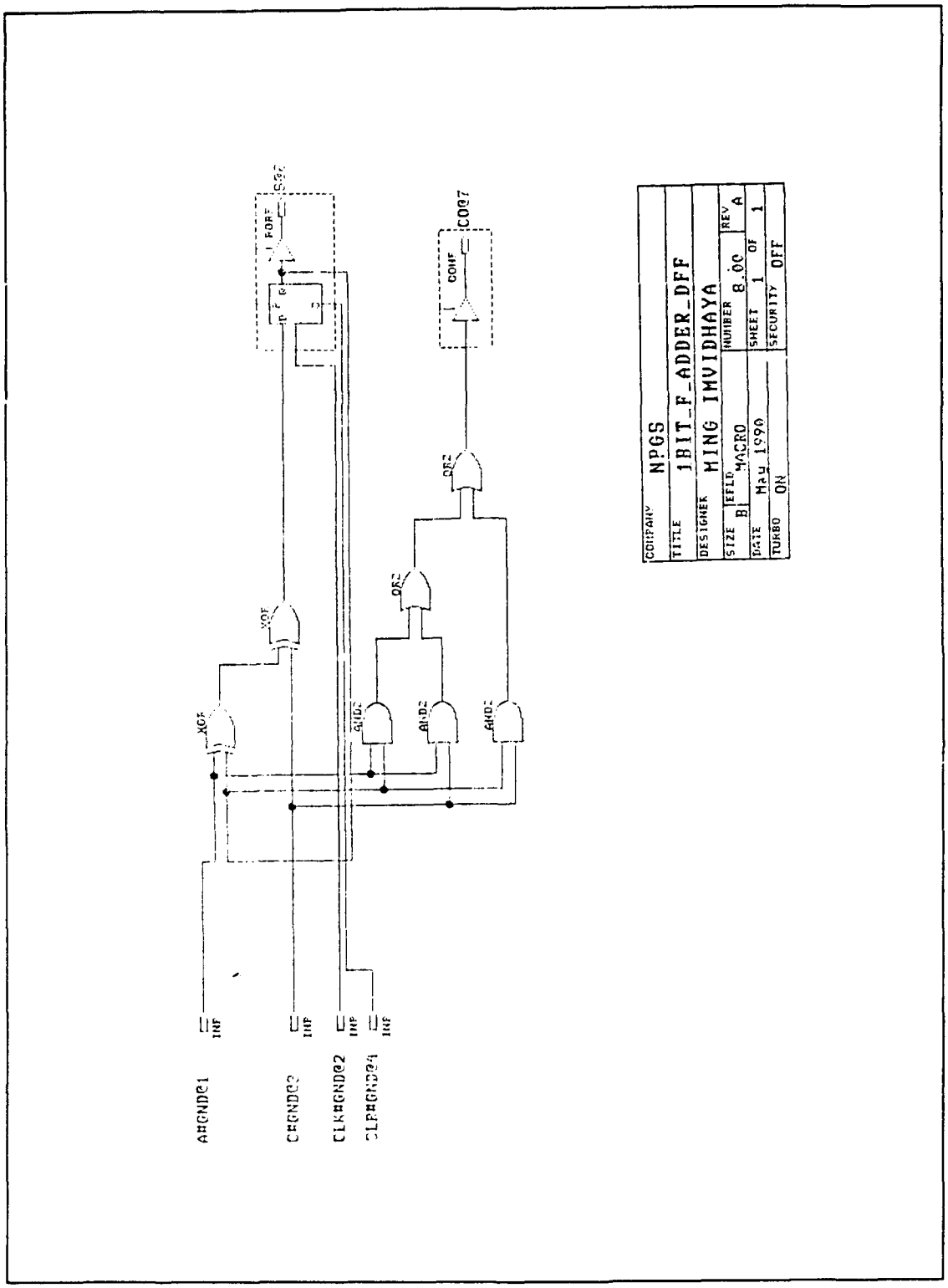


Figure 9. 1-bit full adder with register

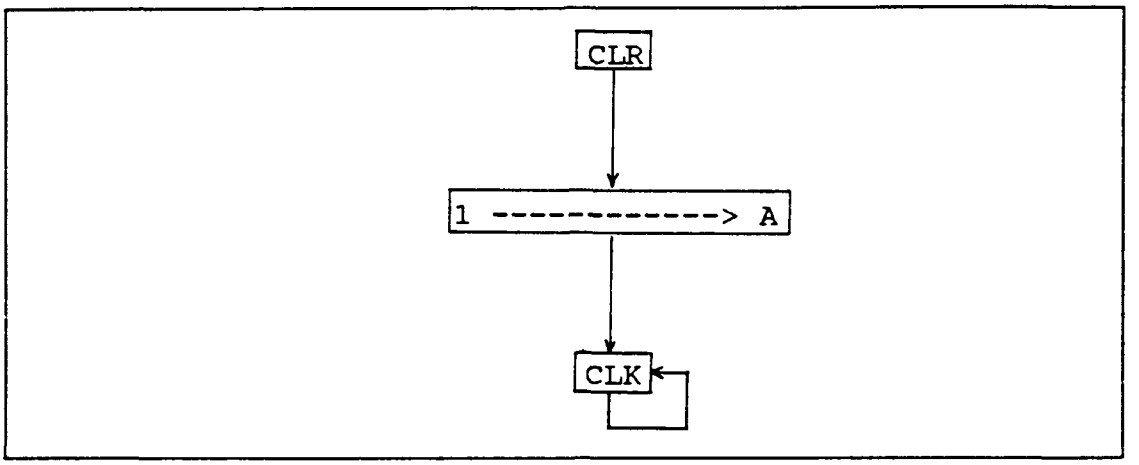


Figure 10. Simple countup flowchart

This circuit of Figure 9 has 4 input signals with initialized value to ground (logic Low). There are 2 outputs SUM and CARRY in the circuit. Every input signal or output signal is assigned a stub number to facilitate the upper level design in ALTERA environment. This circuit of 1-bit full adder with register is put into a library of macrofunctions for later use in upper hierarchy designs.

A 4-bit full adder with register can be built by using the previous macrofunction of 1-bit full adder. Each macrofunctions is connected sequentially as shown in Figure 11. Between each adjacent macrofunction there is an ALTERA primitive called NOCF. This primitive is used to break the product terms of the carry out of the 1-bit macrofunction from that of the other bit so that they are implemented into two different EP1800 macrocells. Otherwise, the ALTERA system will give an error message of "too many p-terms for a single macrocell". Input stubs and output stubs also have the

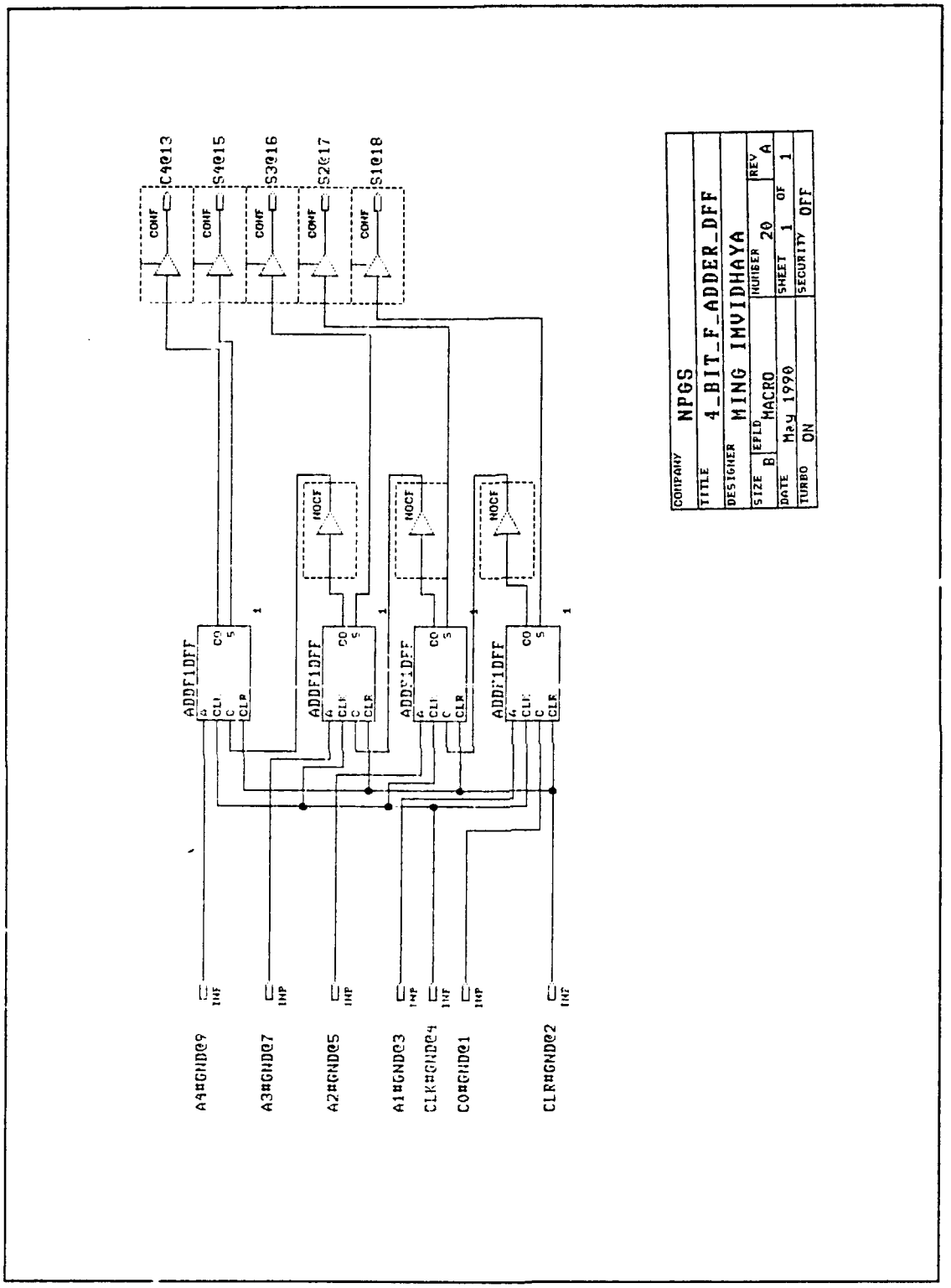


Figure 11. 4-bit full adder with register

initialized value of logic low. The stub number are also shown accordingly. This 4-bit circuit is also put into the macrofunction library.

Finally, a 16-bit full adder with register can be created by using the macrofunctions of the 4-bit adder. The same NOCF primitives are also used to break up the total p-term as mention before. The circuit diagram is shown in Figure 12. In this circuit, the XOR gates are used to complement the input value. The INV signal control this operation. Since the CLR signal in the original design is low activated, an inverter primitive is used to accomplish the active high requirement. A LATCH signal needs to have a clock buffer primitive because this will cause the ALTERA Design Fitter to use a programmable clock pin. Otherwise, an externally connected LATCH signal to the clock pin of the module will be used. Consequently, the LATCH input pin can be reduced from 4 to 1. The clock buffer primitives also impose a longer delay time between the latch input and the clock input of the D-FF. This can ease the setup time requirement for the flip flop and ensure that the D-FF will not be latched before the arrival of the actual data.

F. IMPLEMENTATION OF THE FUNCTION TABLE[i] - TABLE[j]

This function can be performed in hardware by using the circuit module from the previous section and additional control signals from a controller. The sequence of operation can be as follows :

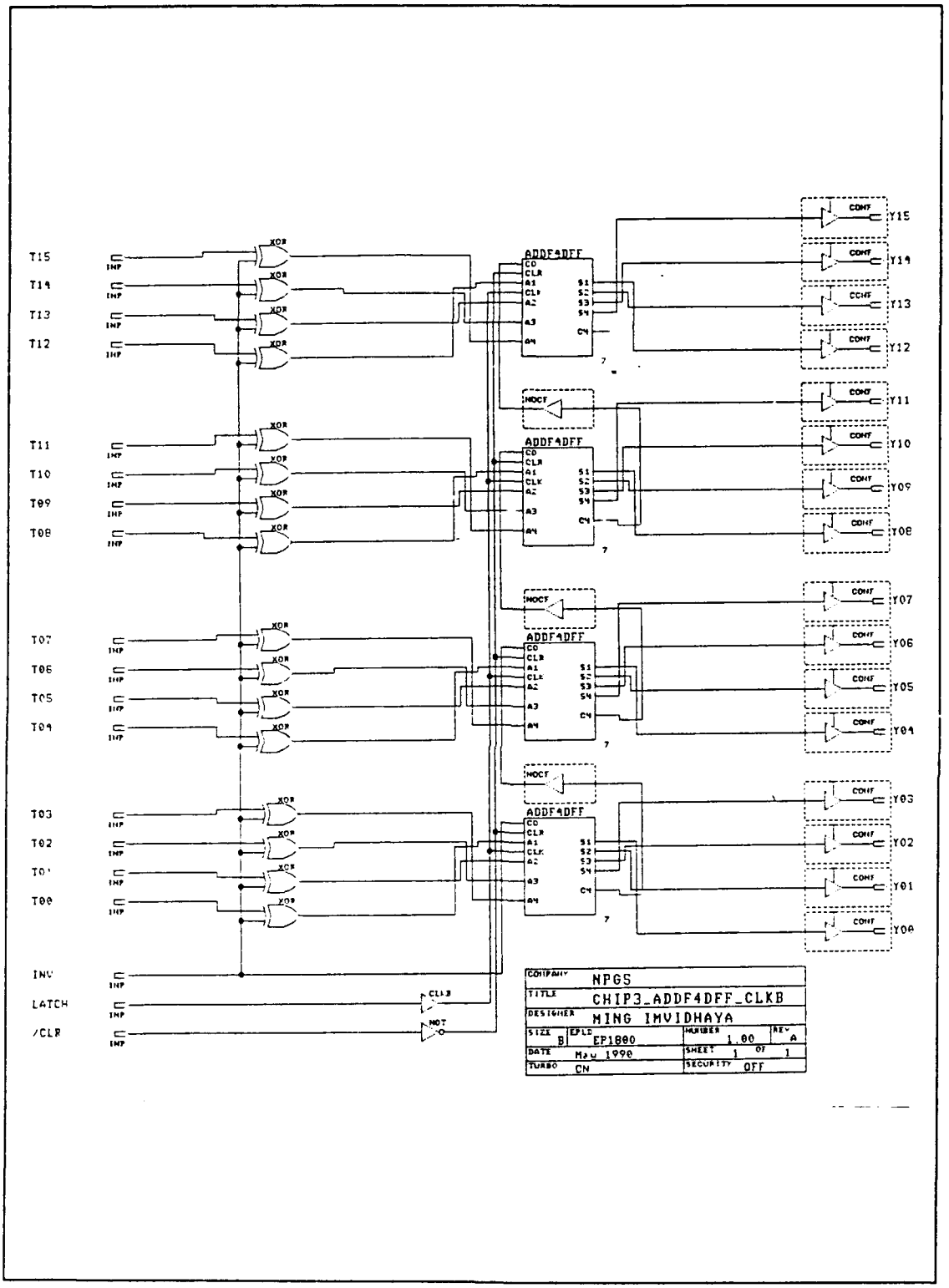


Figure 12. 16-bit full adder with register (CHIP3)

- Clear D-FF by the CLR signal. This will cause A = 0.
- Set B = TABLE[i]. After the LATCH signal, it will cause A = TABLE[i].
- Set B = -TABLE[j]. By using the INV signal to get the two's complement of -TABLE[j]. The LATCH signal will cause A = TABLE[i] - TABLE[j].

The original circuit for this function is shown in Figure 13 which is extracted from the original costfunction design in Figure 2. As mentioned before, using the macrofunction of 4-bit full adder with register, an equivalent circuit is created and shown in Figure 12. This circuit can be integrated into one EP1800 chip in the ALTERA design system. This EPLD device is called CHIP3. The pin assignment for this chip is shown in Figure 14.

The integration of this circuit causes all data paths between the ICs in Figure 13 to be inside one EP1800. This results in reduced space and power consumption. The propagation time is also reduced. Normally, in the original circuit the propagation time is about 150 ns. The tailored EP1800 chip can produce results with a maximum propagation time of 75 ns. The propagation time of the CHIP3 really depends on how ALTERA system assigns the logic expressions to the macrocells. Different implementation of the adder circuits may result in a different propagation time.

Since the gate delay of the original design has no effect in the EPLD implementation, the delay has to be measured from

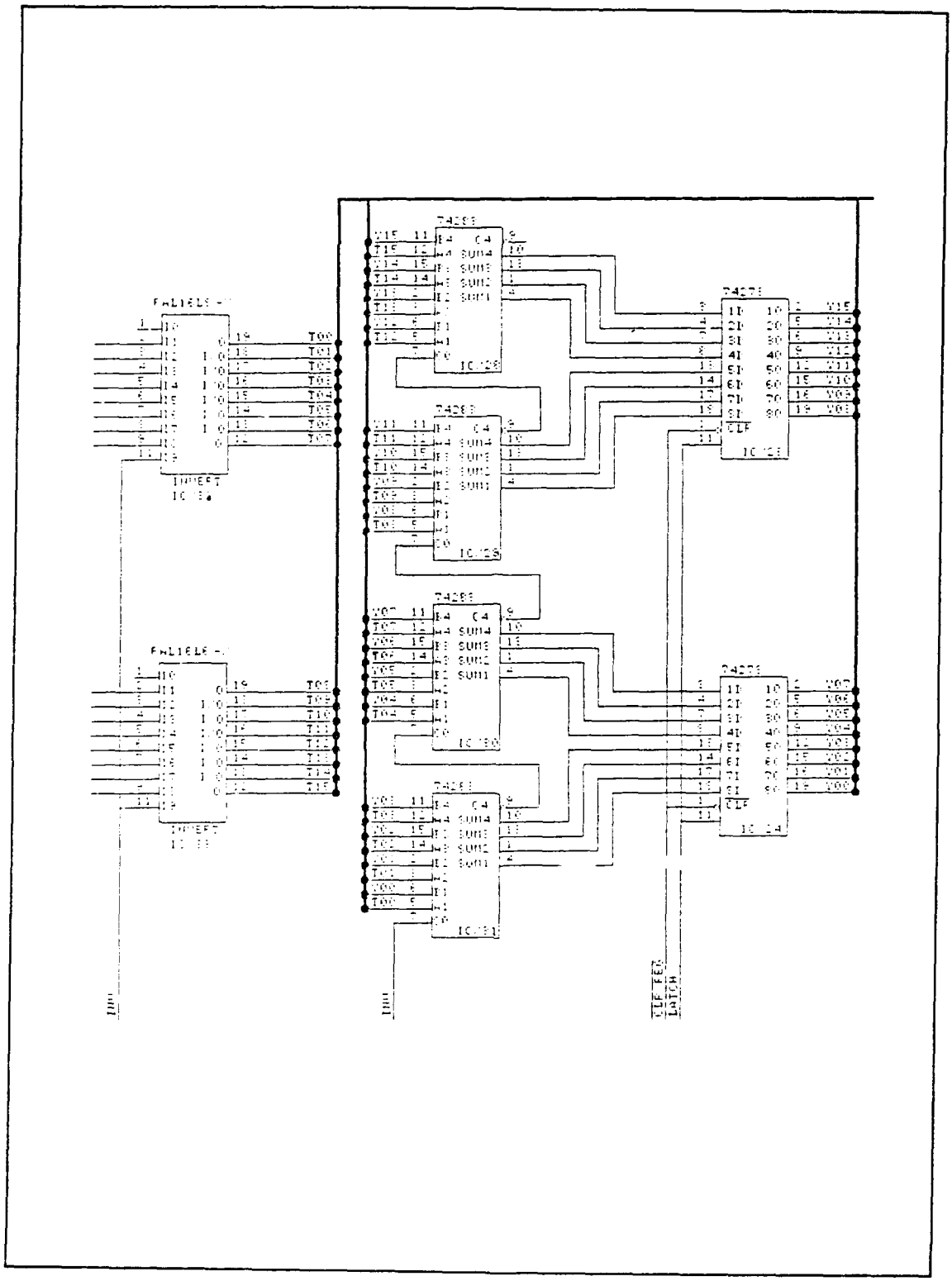


Figure 13. Circuit of TABLE[i] - TABLE[j]

```
ALTERA Design Processor Utilization Report                    ch3dckin.rpt
@(#) FIT Version 6.1    4/4/89 23:56:49 39.16
 ***** Design implemented successfully

MING IMVIDHAYA
NPGS
May 1990
1.00
A
EP1800
CHIP3_ADDF4DFF_CLKB
LogiCaps Schematic Capture Ver 1.9
Input files : ch3dckin.sdf
ADP Options: Minimization = Yes,  Inversion Control = No,  LEF Analysis = No

OPTIONS: TURBO = ON, SECURITY = OFF
```

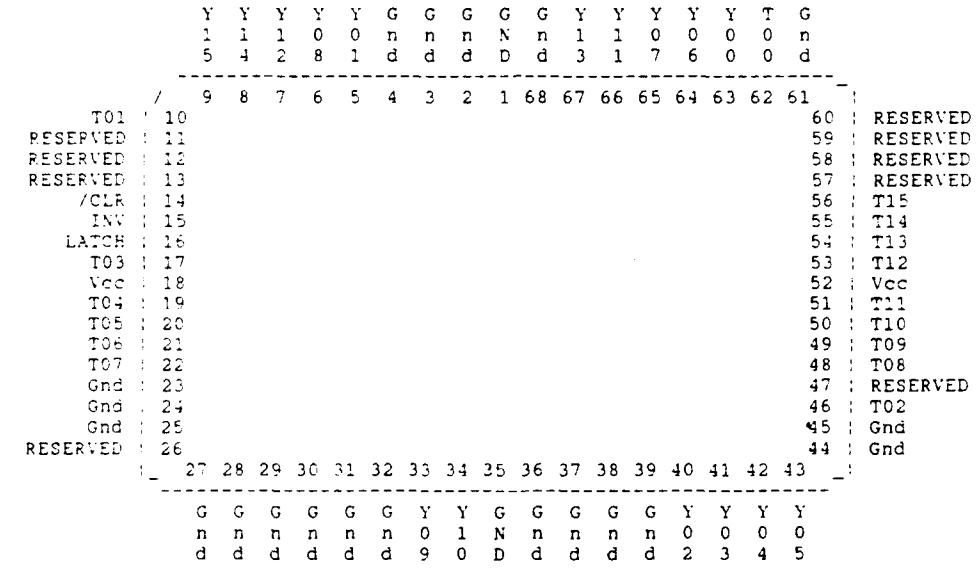


Figure 14. CHIP3 pin assignment

the EP1800 timing model. One disadvantage of using the ALTERA system to configure the macrocell is that the ALTERA system can't guarantee the propagation delay of each output pin to be equal. The reason is due to the different timing of different type of macrocells. For instance, the adder circuit for bit Y10 is assigned to a global macrocell while the adder circuit for bit Y5 is assigned to a local macrocell. This may result in different delay of Y10 and Y5. However, in the worst case when the carry from the least significant bit has to propagate to the most significant bit, the LATCH signal has to wait until the outputs from each adders become stable. After that it is finally possible to gate the result into the registers.

The implementation of this circuit into one EP1800 uses about 90 percent of the total gates of the chip. The idle part of this EPLD may be used to implement another circuit if the remaining number of I/O pins and macrocells are adequate. However, this may cause interference among the assigned pins to the extend that benefit is not usually worth of the trouble.

Signals used in CHIP3 are the following :

- Input data signals: P0 to P15.
- Output data signals: Y0 to Y15.
- INV signal. This signal comes from the control and is used to generate the two's complement of the input data.
- CLR signal. This active low signal is used to clear the output register to zero.

- LATCH signal is a control signal for gating output into the register when the output data become available.

The data from the ACC RAM can also be passed through the CHIP3 to the Y register of the multiplier. This sequence of operations can be shown as follows :

- Assert CLR signal. This causes Y = 0.
- P = ACC[i].
- Assert LATCH signal. This cause Y = ACC[i].

G. IMPLEMENTATION OF THE FUNCTION SIZE[i] - SIZE[j]

This function is originally performed by a circuit in Figure 15. A new circuit is created by using the ALTERA primitives and macrofunctions which is equivalent to the original circuit. This circuit in Figure 16 can be integrated into one EP1800, called CHIP1. Pin assignment of the programmed EP1800 is shown in Figure 17. The control sequence for this operation is the same as those of the previous control operations. The only difference is that this function has only 10 bits input and 12 bits output. That is why the INV signal is connected to the 2 most significant input bits of the adder so that the two's complement of -SIZE[j] for a 12-bit adder will be performed.

The input data, Q (10 bits), comes from the PROM (IC26,27 of Figure 2), and the output data, U, goes into the address register of another PROM (IC16,17 of Figure 2). The required

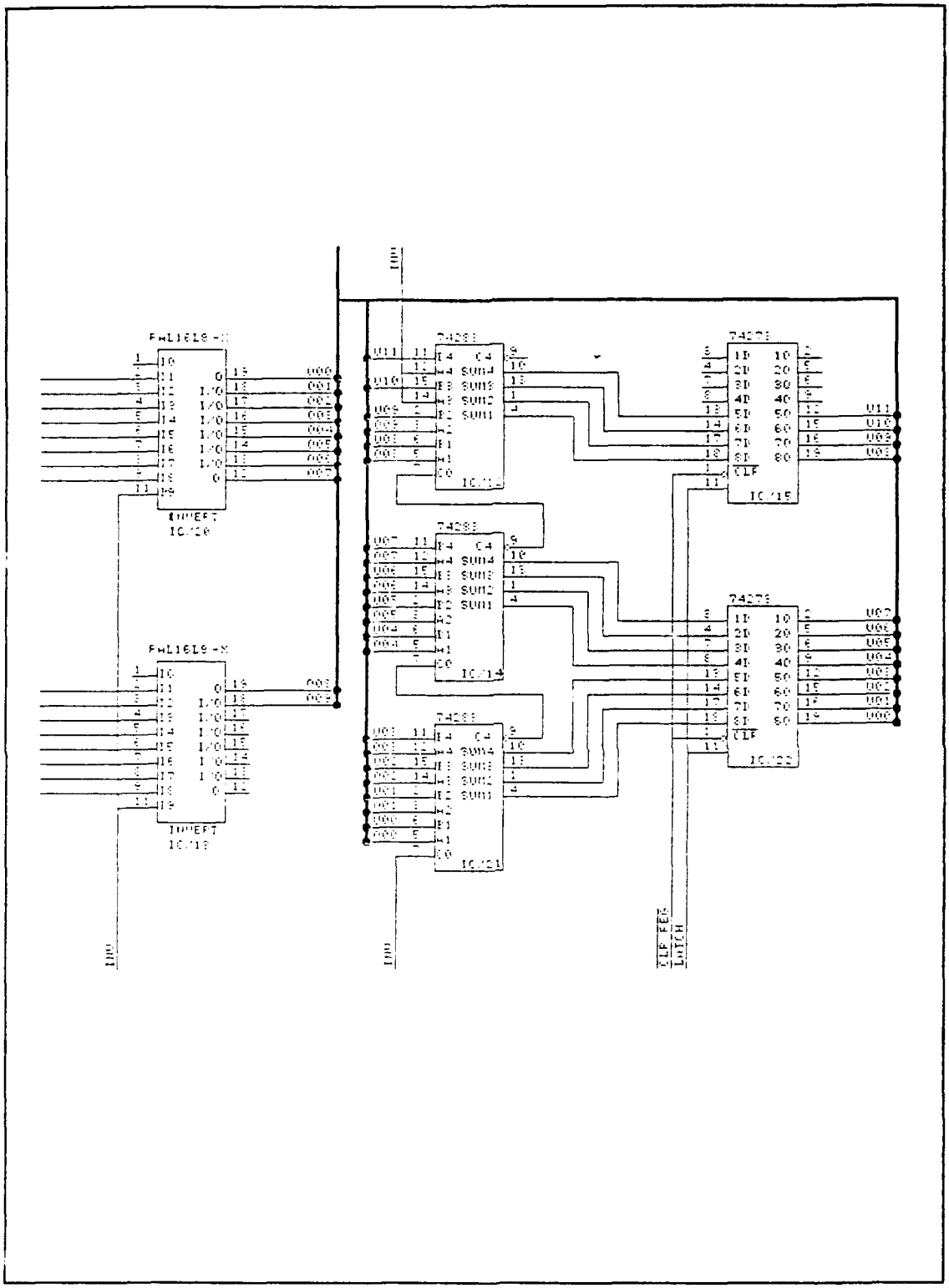


Figure 15. Circuit of SIZE[i] - SIZE[j]

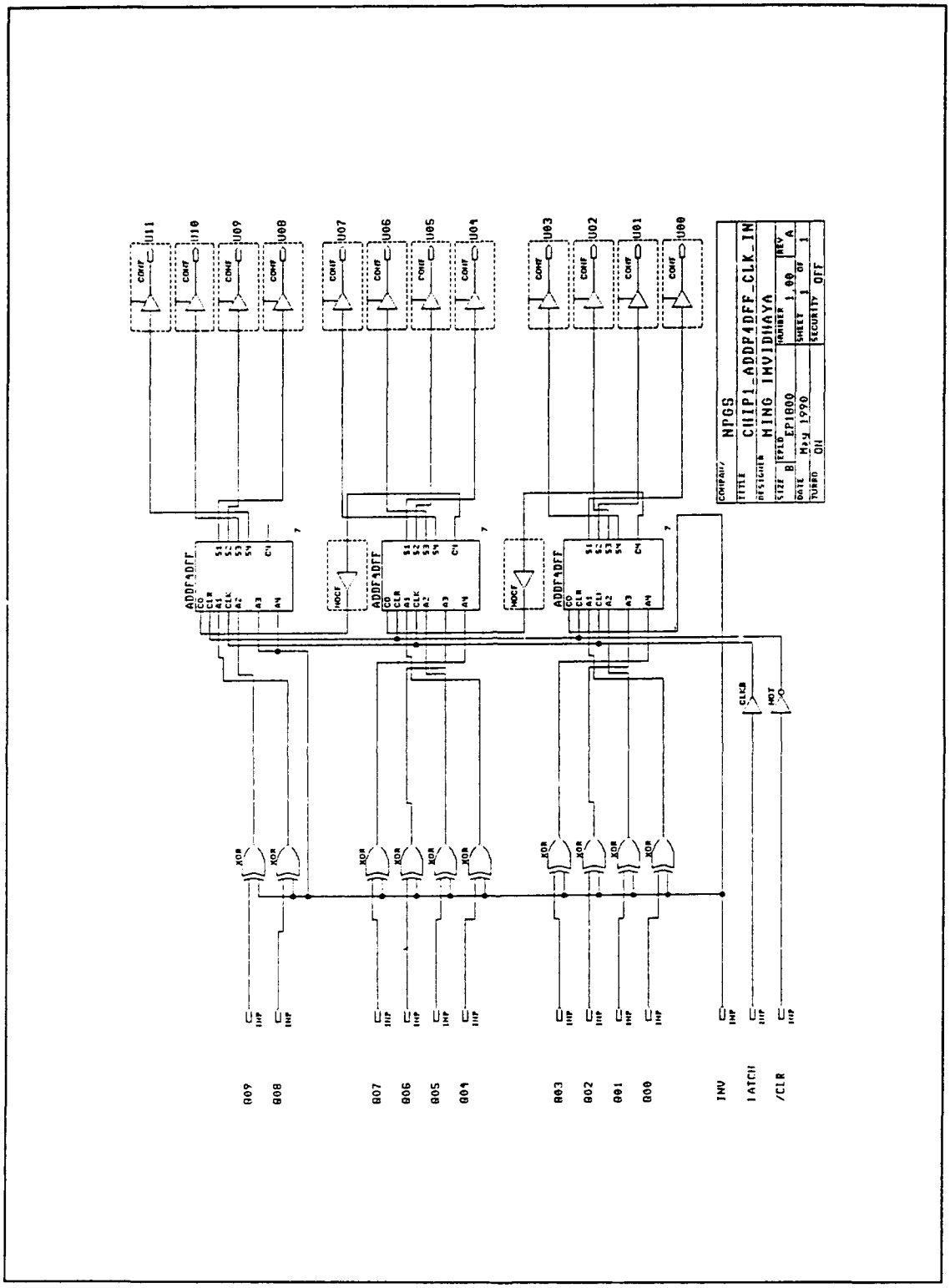


Figure 16. Circuit of CHIP1

```
ALTERA Design Processor Utilization Report                    chldckin.rpt
@(#) FIT Version 6.1    4/4/89 23:56:49 39.16
 ***** Design implemented successfully

MING IMVIDHAYA
NPGS
May 1990
1.00
A
EP1800
CHIP1_ADDF4DFF_CLK_IN
LogiCaps Schematic Capture Ver 1.9
Input files : chldckin.sdf
ADP Options: Minimization = Yes,  Inversion Control = No,  LEF Analysis = No

OPTIONS: TURBO = ON, SECURITY = OFF
```

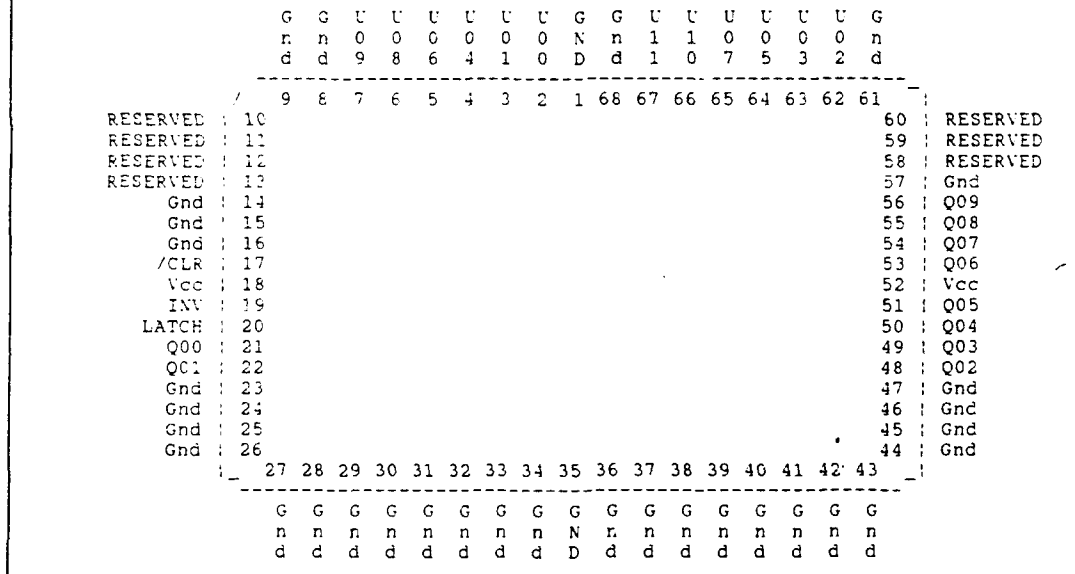


Figure 17. CHIP1 pin assignment

control signals of CHIP1 and CHIP3 are produced from the costfunction state machine, which will be discussed later.

The implementation of this circuit into one EP1800 uses only 20 percent of the total gates. The remaining pins and macrocells may be used to implement other portions of the original design.

H. INPUT BUFFER CIRCUITS

The input buffers accept the values of i and j. The buffer circuits of i and j are shown in Figure 18 and Figure 19 respectively. Two types of TTL IC's used in this circuit are 74ALS574 (octal D-FF) and 74ALS541 (octal line driver).

The implementation of these circuits is created by using the ALTERA primitives. These primitives performs the same function as the TTL IC's above. The circuit is shown in Figure 20. The ALTERA Design System converts this circuit into a JEDEC design file which is programmed into one EP1800 chip. The programmed EP1800 chip for i and j is called CHIP4i and CHIP4j respectively. The pin assignment of this EPLD is shown in Figure 21.

The register of each buffer store 16-bit values of i or j. Thirteen least significant bits are passed from this register to the output. These bits are combine with RAM/MAC signal to create a 14-bit output (M00 to M13). Since values of i and j will be output onto the same bus, the control signals ENI and ENJ will ensure that both values will not be placed on the bus

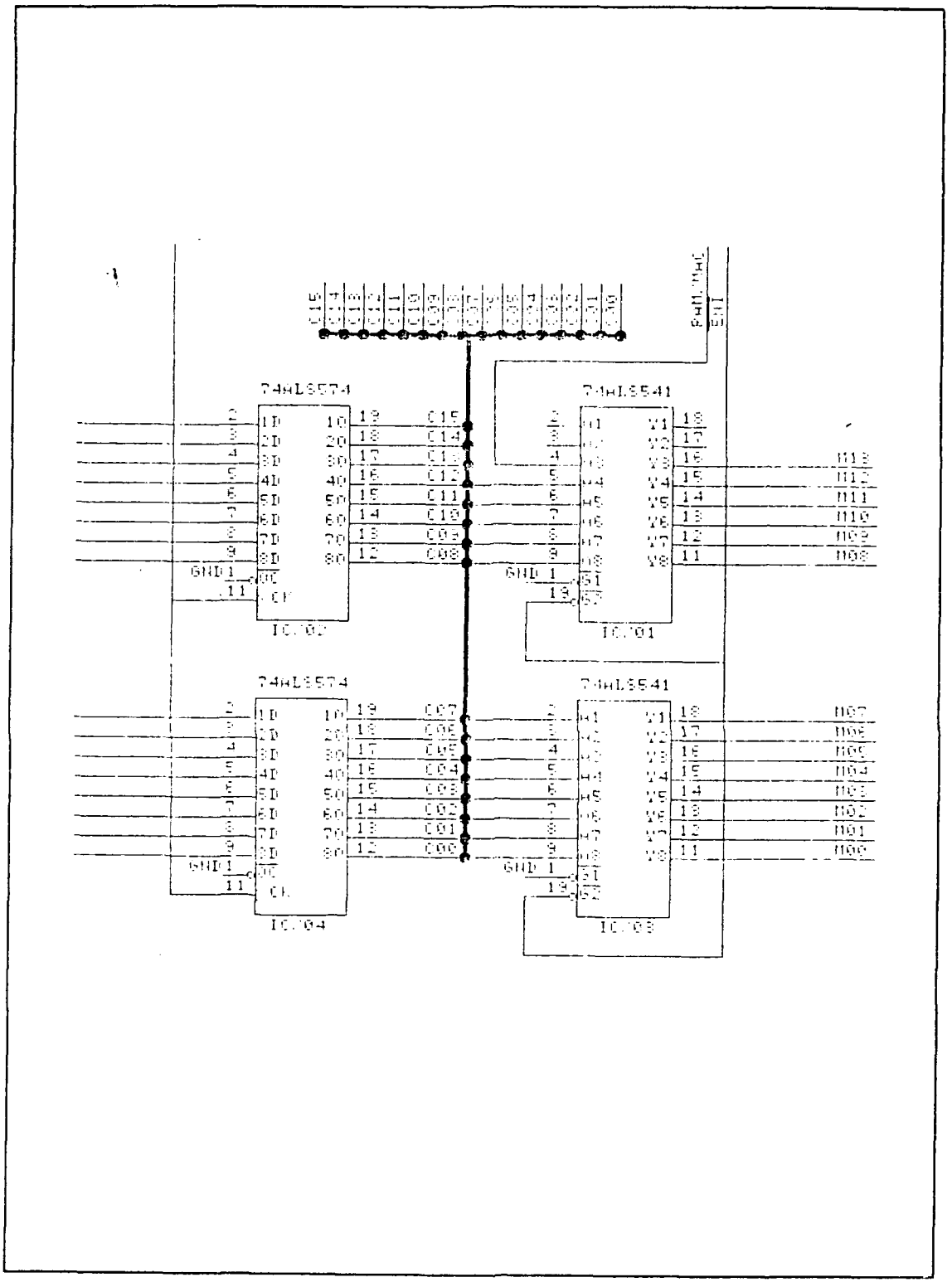


Figure 18. Input buffer i

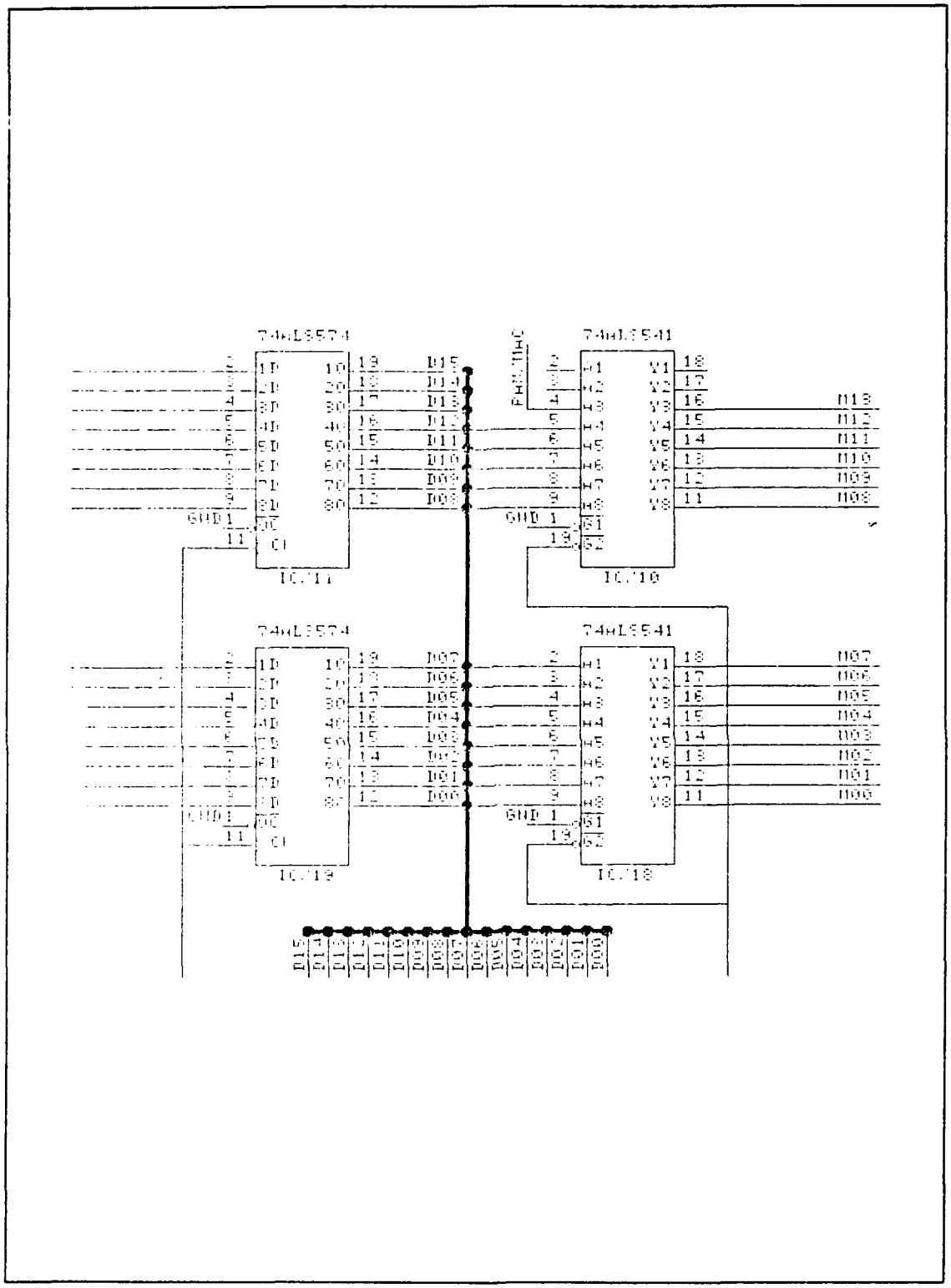


Figure 19. Input buffer j

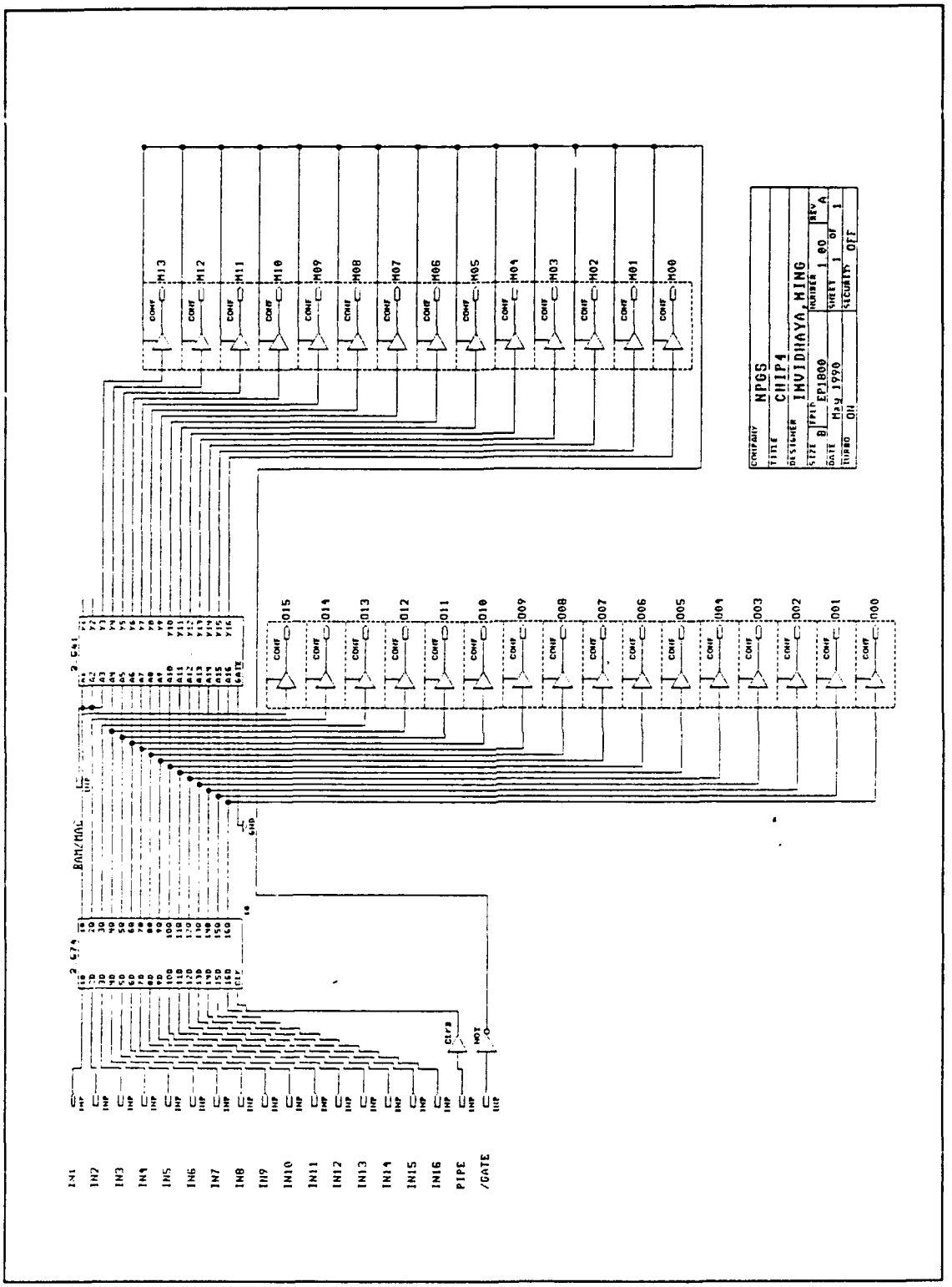


Figure 20. Circuit of CHIP4i and CHIP4j

```
ALTERA Design Processor Utilization Report                          chip4
@(#) FIT Version 6.1    4/4/89 23:56:49 39.16
 ***** Design implemented successfully

IMVIDHAYA,MING
NPGS
May 1990
1.00
A
EP1800
CHIP4
LogiCaps Schematic Capture Ver 1.9
Input files : chip4.sdf
ADP Options: Minimization = Yes,  Inversion Control = No,  LEF Analysis = N

OPTIONS: TURBO = ON, SECURITY = OFF
```

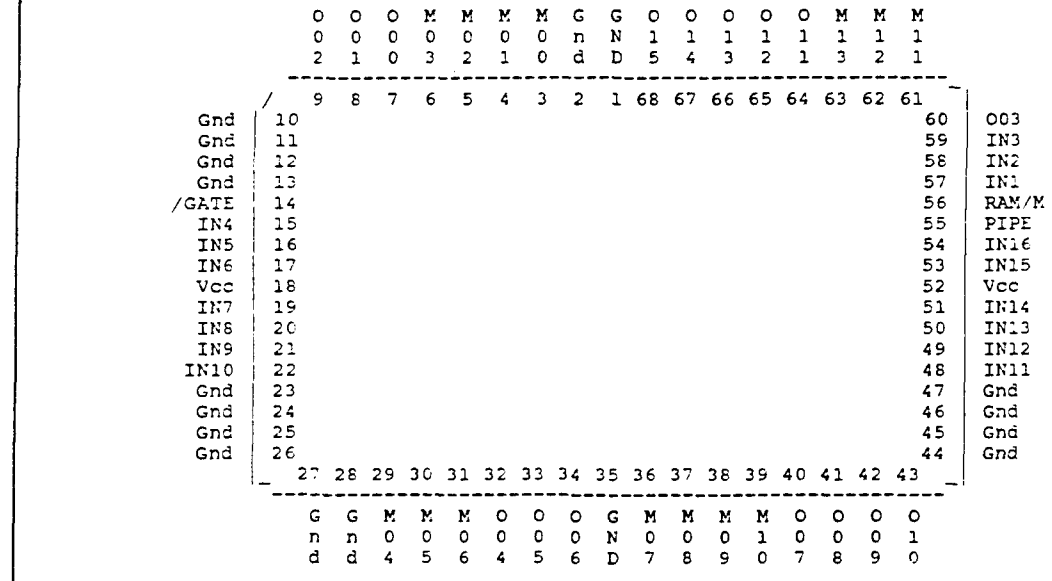


Figure 21. CHIP4 pin assignment

at the same time. Bit M13 (in Figure 2) of the output bus distinguish the accessing between the ACC RAM and the TABLE RAM.

I. CONTROL AND PC-INTERFACE

The costfunction circuit performs the calculation by issuing a sequence of control signals. These control signals are generated according to the state in Figure 2 by IC7. There are a total of 11 states to complete the calculation for each i and j. The state diagram is shown in Figure 7 and its corresponding RTL is shown in Figure 6. A state triggered by the external clock signal will transit to the next state. This clock signal also determines the speed of the costfunction calculation. Initially, the costfunction circuit communicates with the PC-BUS to load the data into the ACC RAM and the TABLE RAM. These can be done via the address decoder (IC40) and the PC-INTERFACE circuit (IC33,39) of Figure 2.

By using the utilities in the ALTERA system, these 2 functions can be combined and implemented into one EP1800. Boolean equations of these two functions are entered in a format of the ALTERA Design File (ADF). This file is shown in Appendix A. The ALTERA Design System converts this file into a JEDEC file which is then used to program the EP1800 hardware.

In the VHDL modeling, a JEDEC file is read into the EP1800 model to tailor the programmed function. From the results of

the simulation, it was shown that the function of the state machine alone can be operated with a clock speed of 20 MHZ. In actual operation, the state machine in the circuit has to wait for the delay of data path elements such as the memory access for the ACC[i], and the TABLE[i], etc. Therefore, the clock speed in the costfunction simulation is slower than is possible with the controller.

The implementation uses 40 percent of the macrocells of one EP1800. The remaining macrocells are reserved for the function of the asynchronous interface module (IC08) whereas the information of the module is not currently available. The rest of the chip may include COUNTER16 (IC09) and PC-DATA bus (IC33,39) if there are enough pins and macrocells available.

The ALTERA system also provides the state table entry instead of the Boolean equation entry. The states of costfunction can be accepted in a format of the State Machine File (SMF) which is created in a text editor. The SMF is converted to ADF by the ALTERA state machine converter. The ADF of the state machine can be merged to the ADF of the PC-interface to produce an ADF of the combined functions. From our experience, more ADF functions can be merged as long as there are enough I/O pins and macrocells available in an EP1800. The ALTERA Design System determines the pins and the macrocells availability. The final result is a JEDEC file which will be used to program the EPLD chip.

The pin assignment of the programmed EP1800 is shown in Figure 22. Signals for state machine are mapped as follows:

- S0 connects to the CLR_REQ signal which is an active low signal.
- S1 connects to the ENABLE signal which is determined by the asynchronous chip to enable the ENi signal or the ENj signal.
- S2 connects to the LATCH signal which is an active high signal.
- S3 connects to the INV signal which is also an active high signal.
- S4 connects to the RAM/MAC signal. The ACC RAM is accessed when the RAM/MAC signal is asserted low otherwise the TABLE RAM is accessed.
- S5 connects to the CLKY signal which is asserted high.
- S6 connects to the CLKX signal which is also asserted high.
- S7 connects to the CF_DONE which is asserted high.

In this chapter a number of circuits of the costfunction are integrated into high density EP1800 chips. The procedures and rationing of partition in the design are discussed. In the next chapter some behavior models used in the final simulation will be discussed.

```
ALTERA Design Processor Utilization Report                          cos_pc.rpt
@(#) FIT Version 6.1     4/4/89 23:56:49 39.16
 ***** Design implemented successfully

MING IMVIDHAYA
NAVAL POST GRADUATE SCHOOL
4/24/90
1.00
B
EP1800
P22V10 COST_FUNCTION & PC_INTERFACE

Input files : cos_pc.adf
ADF Options: Minimization = Yes.  Inversion Control = No,  LEF Analysis = Yes

OPTIONS: TURBO = ON, SECURITY = OFF
```

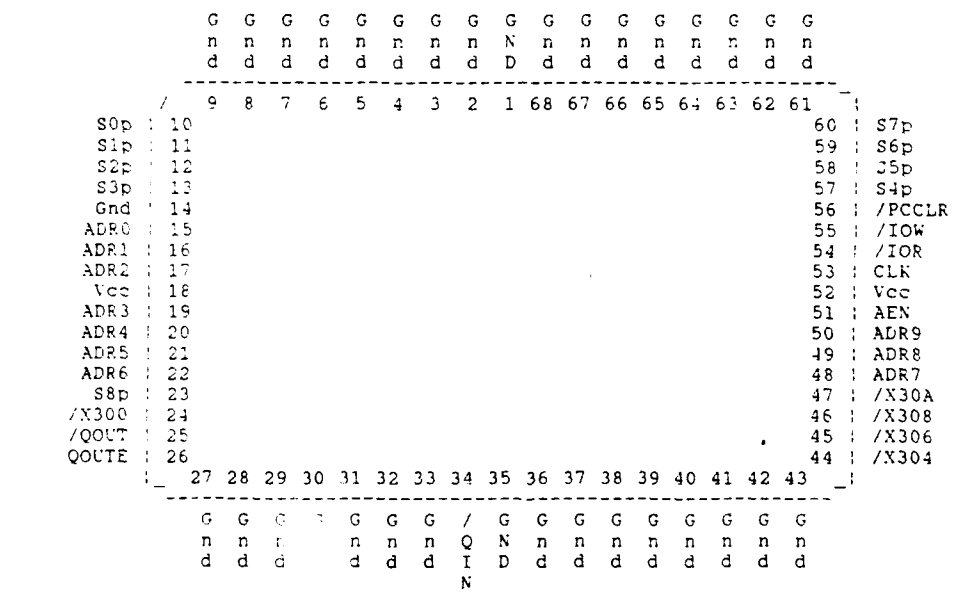


Figure 22. Control and PC-interface pin assignment

III. SOME VHDL BEHAVIOR MODELS

A. VHDL BEHAVIOR MODELING

The modeling in VHDL can be performed either in behavioral level or in structural level. A behavioral model can be defined as the functional interpretation of a particular system. A structural model contains conceptual partitions which decompose the model into functionally related sections. A structural description of a piece of hardware is a description of what its subcomponents are and how the subcomponents are connected to each other. It is an important characteristic of the VHDL that a designer can mix behavioral and structural descriptions at any level [Ref. 9]. This ability to mix description modes offers the designer several advantages. First, the refinement from behavior to structure need not proceed at the same rate for all portions of the design. Therefore, at some stage a design may contain both an abstract behavioral description for unrefined portions and a structural breakdown for portions whose refinement is known. Second, it is not necessary for a designer to simulate everything at a low level design. The portions of the design that have already been verified at a low structural level can be replaced with behavioral versions for incorporation into larger simulations [Ref. 10].

In this section, two VHDL source codes are written as the behavior models of 2 components which are the PROM (IC16,17) and the AM29510 (IC25) in Figure 2. Since the functions of these two components are known, it is easy to write a VHDL code that accepts inputs and produces outputs with the characteristic delays for each component. At this point in time, it is not necessary for the VHDL code to implement the exact hardware architecture of the components. Initially, only simulation of the functionality of the components are concerned.

To implement the VHDL behavior model, a standard logic package from the VANTAGE SYSTEM is used. The VANTAGE SYSTEM is a VHDL support environment. This package includes some functions that can be utilized to convert the binary value into the integer value.

B. 1/SIZE PROM MODELING

The function of this memory circuit, as shown in Figure 23, is to accept a value which is a value of SIZE[i] - SIZE[j] from the CHIP1. This value represents a fixed point number of a fraction. For example, an input value of 25 is interpreted as 0.25 and the output will be 1/0.25 = 4. The PROM also handles a negative number represented in two's complement form.

To implement the function of this PROM in a VHDL behavior model, first the logic input data is converted to an integer

REPRODUCED AT GOVERNMENT EXPENSE

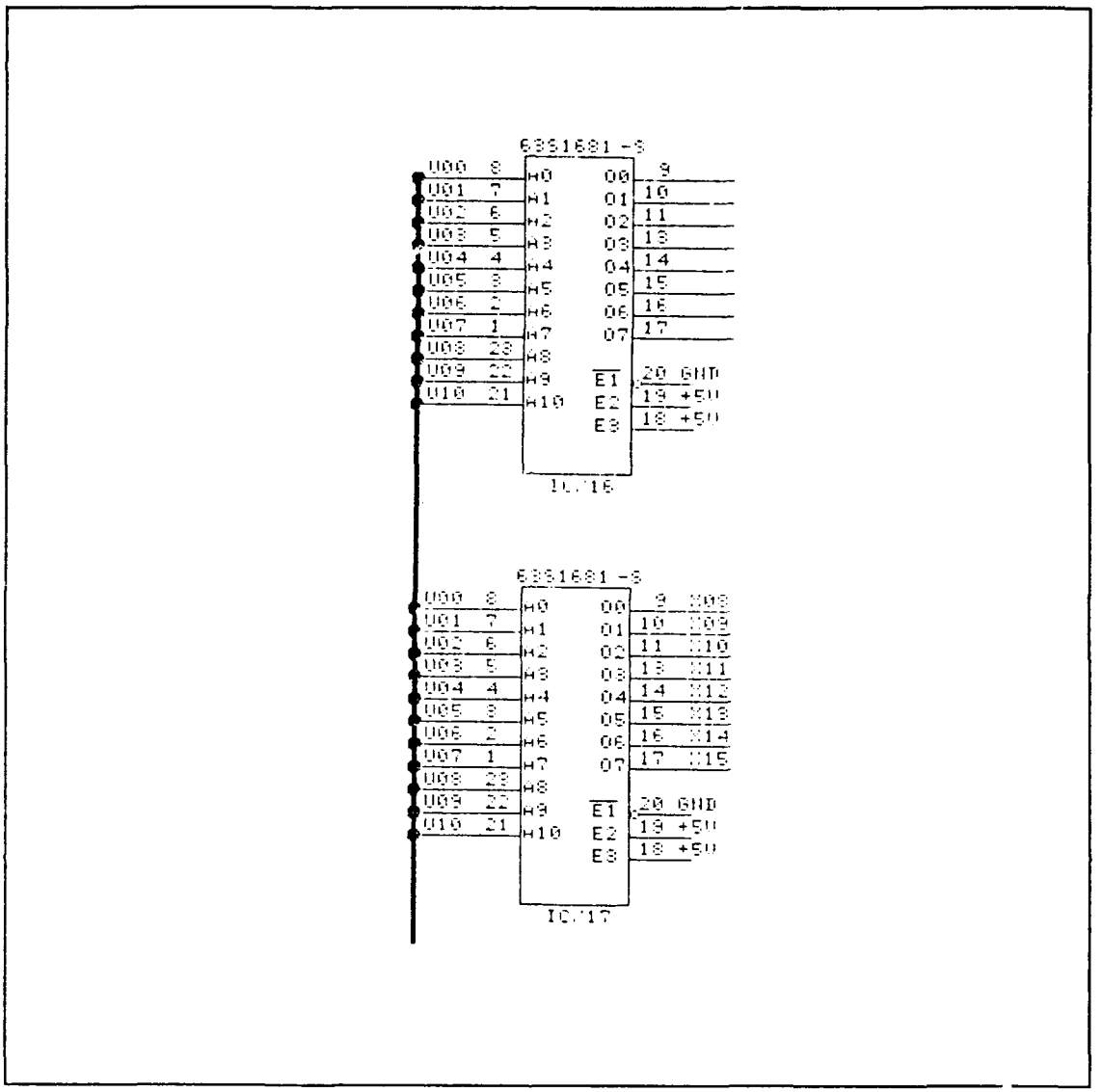


Figure 23. 1/SIZE PROM circuit

value by the subroutine f_logic_to_int2c [Ref. 11]. This integer value is then transformed into a fixed point number. By taking the inverse of this fixed point number, the result is obtained. This result is still in the from of an integer number and is converted to a logic value of 16 bits output by the subroutine f_int_to_logic2c. Since this PROM

handles the inversion of both the positive and the negative values, the VHDL code have to handle two's complement numbers for both input and output. The access time of this PROM is also modeled as 100 ns. The coding of this model is shown in Appendix B.

C. AM29510 MODELING

AM29510 is a high-speed 16x16 bit multiplier/accumulator (MAC) chip. Figure 24 shown the block diagram of this IC. The X and Y input registers can accept 16-bit inputs in either the two's complement or the unsigned magnitude formats. An additional register stores the Two's Complement (TC), Round (RND), Accumulator (ACC), and Substraction/Addition (SUB/ADD) control bits. This register is clocked whenever the X and Y input registers are clocked. The 35-bit accumulator/output register contains the full 32-bit multiplier output which is sign extended or zero-filled based on the TC control bit. The accumulator can also be preloaded from an external source through the bi-directional P port. The operation of the accumulator is controlled by the signals ACC, SUB/ADD, and PREL. Each of the input registers and the output register has independent clocks.

In the VHDL behavior model of the AM29510, some details of this IC are disregarded. For example, the ROUNDING function and the separation of output bits are not simulated. The main purpose of this VHDL coding for the AM29510 is only to support

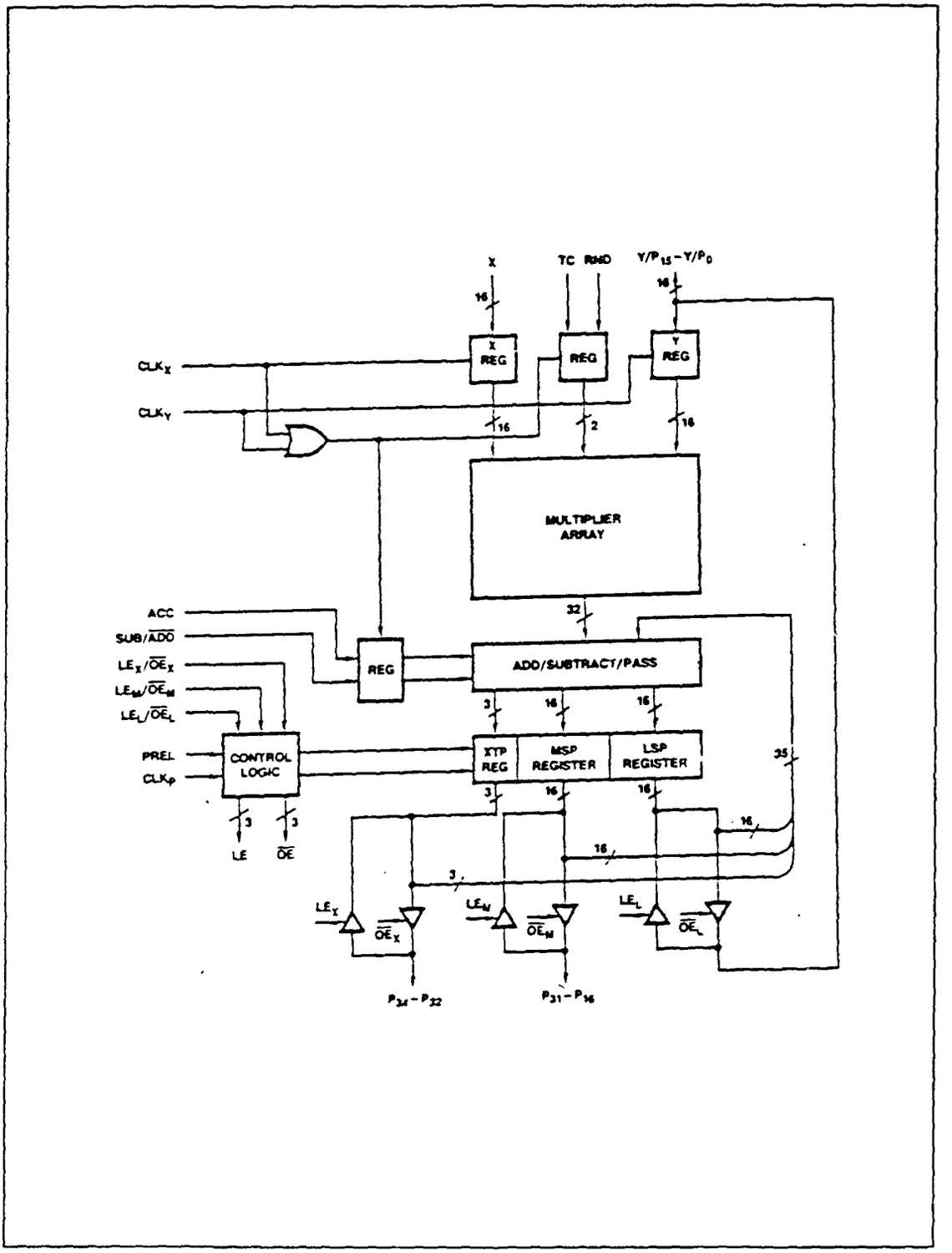


Figure 24. AM29510 block diagram
(adapted from AM29510 DATA SHEET)

the initial simulation of the operations of the costfunction. The program, accepts two 16-bit inputs and produces one 16-bit output. In the model studied here only the necessary control signals are implemented. This excludes RND, and PREL signals. The LEM/OEM, LEX/OEX, and LEL/OEL signals are combined into one LE/OE signal. The model is divided into two portions. The first portion handles the timing characteristic of the AM29510. The second portion handles the functions of the multiplication and the accumulation. Appendix C includes the VHDL coding of the AM29510.

D. SUMMARY

The behavioral modeling done in this research offers the capability to simulate the initial function requirements of the costfunction circuit. An important fact for each model is that the result is reasonable within the specified delay. The behavioral model that is effi ent can help a fast simulation in high-level design. At the time when the structure models are created, the behavioral models can be replaced by the structural models directly. The simulation of a behavioral description is usually less time consuming than a structural description. But, a structural model may provide a more accurate description of the actual hardware operations in a chip. Therefore, both the behavior model and the structure model have advantages and disadvantages in applications.

IV. VHDL SIMULATION AND ANALYSIS

A. THE SIMULATION OF COMBINED CIRCUIT

In this chapter, implementation of the circuits in the previous chapter are combined and simulated in VHDL. The observation is concentrated on the operations of CHIP3, CHIP1, 1/SIZE PROM, and AM29510 by invoking control signals from the CONTROL chip. A VHDL testbench program is created for the testing [Ref. 12], which is in Appendix D. This program includes all the partition circuit components. There is a process to handle the values supplied from RAM to CHIP3 and CHIP1. The program uses the EP1800 structural model, as developed by a previous NPS thesis student. At the simulation time, each EP1800 is personalized by reading a particular JEDEC file for its designed functions. There are 3 JEDEC files to be read for CHIP3, CHIP1, and CONTROL modules respectively. Names are also assigned to each EP1800 pin corresponding to the pin mapping report file from the ALTERA system.

When the simulation starts, the necessary signals and variables are assigned to their initial values. Then, the PC_CLR signal is asserted in order to reset the state machine to state 0. The present state is changed to the next state after the rising edge of the CLK signal. The final state is state 9. In this state, the result is obtained at the output

of the AM29510 together with the CF_DONE signal. The result is available correctly up to 90 nanoseconds after the rising edge of the CF_DONE signal. The simulation result is shown in Appendix E. The testing values for the simulation is as follows :

	Binary	Decimal
TABLE[i]	= 0000000000001111;	15
TABLE[j]	= 0000000000000001;	1
SIZE[i]	= 000000001111 ;	15
SIZE[j]	= 000000000001 ;	1
ACC[i]	= 0000000000000010;	2

The result of COSTFUNCTION[i,j] = 0000000001100000, which is translated to 96. The actual result should be 98.

Since the content of the 1/SIZE PROM is not available to us at the present time, a simple mathematical inversion was assumed. The VHDL PROM implementation uses integer calculations internally. This should cause a slight difference between the VHDL simulated result and the anticipated PROM result. However, the simulation produces a close enough result to the hardware implementation.

B. TIMING DELAY ANALYSIS

The timing characteristics of each EP1800 depends on how the EPLD is programmed. Although, the EP1800 does not have individual gate delay timing, there still are delays associated with the macrocell, the feedback path, as well as

the I/O control section. In order to operate the circuit correctly, the state machine (CONTROL) must produce each state control signal at the appropriate time after the longest delay of the data path in EPLDs elapses. In the simulation, the measurement of CHIP3 and CHIP1 is made. The longest delay time from input to output is 1200 nanoseconds.

Most of the delay occurs in CHIP3 and CHIP1. These two EPLDs implement the circuit of the 16 bits and the 12 bits full adder with registers. The worst case delay is caused by the ripple carry propagation from the least significant bit to the most significant bit. The EP1800 has 3 types of macrocell which has difference delay characteristics. Since different bit in the full adder may be programmed into difference type of macrocell, this result in the different delay in the output bits. The CONTROL has to ensure that the output signal is stable and then latch the result and change the state.

C. CIRCUIT SPEED ANALYSIS

In this simulation, the clock speed for the state machine is varied experimentally. A suitable clock period is 300 nanoseconds. Compared to the original design with a clock rate of 25 nanoseconds, this implementation is considerably slower. This is a trade off in the design for high integration. However, the board space, as well as power consumption, was reduced. The design can also be changed by re-programming the

EPLD. The EPLD can also be replaced by a faster version of the EPLD in the future.

D. IMPROVEMENT

There are several ways that can improve the efficiency of the integrated implementation. One is to redesign the adder circuit. A carry-look-ahead adder circuit may be desirable and can be easily designed. Another way is to use the idle macrocells and logic for more integration. The added integration may include some functions from the upper level circuitry. The efficiency can also be improved if the EPLD pin assignment can be manually arranged so that the delay between macrocells is minimum. However, the side effect is inconvenience and time consuming. The simulation time of the testbench takes almost 4 hours to complete. For the full implementation of the costfunction PCB, the simulation time may take 6 to 8 hours. It is expected that in a faster computer system the simulation time can be greatly reduced.

V. CONCLUSIONS

A. CONCLUSIONS AND FUTURE RESEARCH

In this research, a software costfunction routine is converted into a Register Transfer Language (RTL) description. This RTL description can be used to design the hardware. The original costfunction circuit in TTL logic is also redesigned to achieve a high integration using EP1800s. The integration is based on the partition of the costfunction circuit into several modules, each of which corresponds to a statement in the costfunction software. A VHDL structural model of EP1800 is used to simulate the function modules of CHIP3, CHIP1, CONTROL, CHIP4i, and CHIP4j. An ALTERA JEDEC file is read into the model to personalize the functions. The PROMs and the AM29510 are also described by their behavior models in VHDL in the final simulation experiment.

The total simulation for the major operation of the costfunction circuit is done at the end. The result is slightly different from the anticipated value due to the integer calculation in the 1/SIZE PROM VHDL model. This can be improved if more information for this PROM can be obtained. The speed of the redesigned costfunction circuit is its main drawback due to the long delay of the EPLDs. The total

simulation turn-around time on a Micro VAX is also discouraging.

For future research, it is desirable to simulate the remaining part of the original design, including the pipeline and sequence generator. After that, the graph partition hardware is the final target for the total simulation. Since there are many unused macrocells left in the revised costfunction circuit, these can be used to integrate higher level functions. It is also hoped that the future technology of EPLD can provide a reduced delay time for the device. At that time, the simulation of this revised design will produce better results.

APPENDIX A - CONTROL AND PC INTERFACE

```
MING IMVIDHAYA
NAVAL POST GRADUATE SCHOOL
4/24/90
1.00
B
EP1800
P22V10 COST_FUNCTION & PC_INTERFACE

OPTIONS:TURBO=ON,SECURITY=OFF
PART:EP1800
INPUTS: CLK,/PCCLR,
        ADR9,ADR8,ADR7,ADR6,ADR5,ADR4,ADR3,ADR2,ADR1,ADR0,
        /IOR,/IOW,AEN,/QIN
OUTPUTS:S8p@23,S7p@60,S6p@59,S5p@58,S4p@57,S3p@13,S2p@12,
        S1p@11,S0p@10,
        /X304@44,/X306@45,/X308@46,/X30A@47,/X300@24,
        /QOUT@25,QOUTE@26

NETWORK:

CLK = INP(CLK)
CLOCK = CLKB(CLK)
/PCCLR = INP(/PCCLR)
PCCLR = NOT(/PCCLR)
S8p,S8  = RORF(S8c,CLOCK,GND,GND,VCC)
S7p,S7  = RORF(S7c,CLOCK,GND,GND,VCC)
S6p,S6  = RORF(S6c,CLOCK,GND,GND,VCC)
S5p,S5  = RORF(S5c,CLOCK,GND,GND,VCC)
S4p,S4  = RORF(S4c,CLOCK,GND,GND,VCC)
S3p,S3  = RORF(S3c,CLOCK,GND,GND,VCC)
S2p,S2  = RORF(S2c,CLOCK,GND,GND,VCC)
S1p,S1  = RORF(S1c,CLOCK,GND,GND,VCC)
S0p,S0  = RORF(S0c,CLOCK,GND,GND,VCC)
ADR9 = INP(ADR9)
ADR8 = INP(ADR8)
ADR7 = INP(ADR7)       nADR7 = NOT(ADR7)
ADR6 = INP(ADR6)       nADR6 = NOT(ADR6)
ADR5 = INP(ADR5)       nADR5 = NOT(ADR5)
ADR4 = INP(ADR4)       nADR4 = NOT(ADR4)
ADR3 = INP(ADR3)       nADR3 = NOT(ADR3)
ADR2 = INP(ADR2)       nADR2 = NOT(ADR2)
ADR1 = INP(ADR1)
ADR0 = INP(ADR0)       nADR0 = NOT(ADR0)
nIOR = INP(/IOR)       IOR = NOT(nIOR)
nIOW = INP(/IOW)       IOW = NOT(nIOW)
```

```
AEN  = INP(AEN)          nAEN = NOT(AEN)
nQIN = INP(/QIN)         QIN = NOT(nQIN)
QOUTEc = AND(IOR,nAEN,ADR9,ADR8,nADR7,nADR6,
             nADR5,nADR4,nADR3,nADR2,ADR1,nADR0)
QOUTE = CONF(QOUTEc,VCC)
/X304 = CONF(nX304c,ADR9)   nX304c = NOT(X304)
/X306 = CONF(nX306c,ADR9)   nX306c = NOT(X306)
/X308 = CONF(nX308c,ADR9)   nX308c = NOT(X308)
/X30A = CONF(nX30Ac,ADR9)   nX30Ac = NOT(X30A)
/X300 = CONF(nX300c,ADR9)   nX300c = NOT(X300)
/QOUT = CONF(nQOUTc,QOUTEc)  nQOUTc = NOT(QOUT)

EQUATIONS:

S8c = /PCCLR*/S8*/S7*/S6*/S5*/S4*/S3*/S2*S1*S0 +
      /PCCLR*/S8*/S7*/S6*S5*S4*/S3*/S2*S1*/S0 +
      /PCCLR*/S8*/S7*/S6*/S5*S4*/S3*S2*/S1*S0;

S7c = /PCCLR*/S8*/S7*/S6*/S5*/S4*/S3*/S2*/S1*/S0;

S6c = /PCCLR*S8*/S7*/S6*/S5*/S4*/S3*/S2*S1*S0 +
      /PCCLR*/S8*/S7*/S6*/S5*S4*S3*S2*/S1*S0;

S5c = /PCCLR*/S8*/S7*S6*/S5*/S4*/S3*S2*S1*S0 +
      /PCCLR*/S8*/S7*/S6*/S5*S4*S3*S2*/S1*S0 +
      /PCCLR*/S8*/S7*/S6*/S5*/S4*/S3*/S2*/S1*/S0;

S4c = /PCCLR*/S8*/S7*S6*/S5*/S4*/S3*S2*S1*S0 +
      /PCCLR*/S8*/S7*/S6*S5*S4*/S3*/S2*S1*/S0 +
      /PCCLR*S8*/S7*/S6*/S5*S4*/S3*/S2*S1*S0 +
      /PCCLR*/S8*/S7*/S6*/S5*S4*S2*/S1*S0 +
      /PCCLR*S8*/S7*/S6*/S5*S4*S3*/S2*/S1*S0;

S3c = /PCCLR*/S8*/S7*/S6*/S5*S4*/S3*S2*/S1*S0 +
      /PCCLR*S8*/S7*/S6*/S5*S4*S3*/S2*/S1*S0;

S2c = /PCCLR*S8*/S7*/S6*/S5*/S3*/S2*S1*S0 +
      /PCCLR*S8*/S7*/S6*/S5*S4*S3*/S2*/S1*S0;

S1c = PCCLR +
      /S7*/S6*/S5*/S4*/S3*/S2*S1*S0 +
      /S8*/S7*S6*/S5*/S4*/S3*S2*S1*S0 +
      /S8*/S7*/S6*S5*S4*/S3*/S2*S1*/S0 +
      /S8*S7*/S6*S5*/S4*/S3*/S2*/S1*/S0;

S0c = PCCLR +
      /S7*/S6*/S5*/S4*/S3*/S2*S1*S0 +
      /S8*/S7*/S6*S5*S4*/S3*/S2*S1*/S0 +
      S8*/S7*/S6*/S5*/S3*/S2*S1*S0 +
      /S8*/S7*/S6*/S5*S4*/S3*S2*/S1*S0 +
      S8*/S7*/S6*/S5*S4*S3*/S2*/S1*S0 +
```

```
      /S8*S7*/S6*S5*/S4*/S3*/S2*/S1*/S0;

X300 = IOW*/AEN*ADR9*ADR8*/ADR7*/ADR6*/ADR5*/ADR4*
      /ADR3*/ADR2*/ADR1*/ADR0;
X304 = IOW*/AEN*ADR9*ADR8*/ADR7*/ADR6*/ADR5*/ADR4*
      /ADR3*ADR2*/ADR1*/ADR0;
X306 = IOW*/AEN*ADR9*ADR8*/ADR7*/ADR6*/ADR5*/ADR4*
      /ADR3*ADR2*ADR1*/ADR0;
X308 = IOW*/AEN*ADR9*ADR8*/ADR7*/ADR6*/ADR5*/ADR4*
      ADR3*/ADR2*/ADR1*/ADR0;
X30A = IOR*/AEN*ADR9*ADR8*/ADR7*/ADR6*/ADR5*/ADR4*
      ADR3*/ADR2*ADR1*/ADR0;
QOUT = QIN;

END$
```

APPENDIX B - 1/SIZE PROM BEHAVIOR MODEL

```
-- SIZEPROM.VHD
library SHU;
USE SHU.EPROM_PACK.ALL;
USE std.std_logic.ALL;
USE std.std_ttl.ALL;
USE work.intpack_ttl.ALL;
USE work.mingpack.ALL;
ENTITY sizeprom IS
    PORT    (x0,x1,x2,x3,x4,x5,x6,x7,
             x8,x9,x10,x11,x12,x13,x14,x15 : IN tri_state;
             y0,y1,y2,y3,y4,y5,y6,y7,
             y8,y9,y10,y11,y12,y13,y14,y15 : OUT tri_state);
END sizeprom;

ARCHITECTURE full OF sizeprom IS
CONSTANT tac :TIME := 100 ns;
SIGNAL   xx0,xx1,xx2,xx3,xx4,xx5,xx6,xx7,
         xx8,xx9,xx10,xx11,xx12,xx13,xx14,xx15,
         yy0,yy1,yy2,yy3,yy4,yy5,yy6,yy7,
         yy8,yy9,yy10,yy11,yy12,yy13,yy14,yy15 : t_wlogic;
BEGIN
    -- input delay processing
    xx0 <= f_ttl(tri_to_tstate(x0)) AFTER tac;
    xx1 <= f_ttl(tri_to_tstate(x1)) AFTER tac;
    xx2 <= f_ttl(tri_to_tstate(x2)) AFTER tac;
    xx3 <= f_ttl(tri_to_tstate(x3)) AFTER tac;
    xx4 <= f_ttl(tri_to_tstate(x4)) AFTER tac;
    xx5 <= f_ttl(tri_to_tstate(x5)) AFTER tac;
    xx6 <= f_ttl(tri_to_tstate(x6)) AFTER tac;
    xx7 <= f_ttl(tri_to_tstate(x7)) AFTER tac;
    xx8 <= f_ttl(tri_to_tstate(x8)) AFTER tac;
    xx9 <= f_ttl(tri_to_tstate(x9)) AFTER tac;
    xx10 <= f_ttl(tri_to_tstate(x10)) AFTER tac;
    xx11 <= f_ttl(tri_to_tstate(x11)) AFTER tac;
    xx12 <= f_ttl(tri_to_tstate(x12)) AFTER tac;
    xx13 <= f_ttl(tri_to_tstate(x13)) AFTER tac;
    xx14 <= f_ttl(tri_to_tstate(x14)) AFTER tac;
    xx15 <= f_ttl(tri_to_tstate(x15)) AFTER tac;


    -- operation

          PROCESS(xx0,xx1,xx2,xx3,xx4,xx5,xx6,xx7,
                  xx8,xx9,xx10,xx11,xx12,xx13,xx14,xx15)
             VARIABLE state : t_logarray(1 TO 16);
```

```
      VARIABLE x : integer := 0;
      VARIABLE xr: real    := 0.0;
      VARIABLE y : integer := 0;
      VARIABLE p : integer := 0;
      VARIABLE unknown : boolean := true;

   BEGIN

     -- pickup x data

            state(16) := xx0;
            state(15) := xx1;
            state(14) := xx2;
            state(13) := xx3;
            state(12) := xx4;
            state(11) := xx5;
            state(10) := xx6;
            state(9)  := xx7;
            state(8)  := xx8;
            state(7)  := xx9;
            state(6)  := xx10;
            state(5)  := xx11;
            state(4)  := xx12;
            state(3)  := xx13;
            state(2)  := xx14;
            state(1)  := xx15;

            f_logic_to_int2c(state,unknown,x);

         xr := REAL(x);
         WHILE xr > 1.0 LOOP
               xr := xr*0.1;
         END LOOP;

if xr = 0.0 then
    unknown := true;
else
      x := INTEGER(1.0/xr);
end if;

   IF NOT unknown THEN
      f_int1_to_logic2c(x,state,ttl);
END IF;

     -- assign values to output signals
   IF NOT unknown THEN
               yy0 <= state(16);
               yy1 <= state(15);
               yy2 <= state(14);
               yy3 <= state(13);
               yy4 <= state(12);
```

```
                        yy5 <= state(11);
                        yy6 <= state(10);
                        yy7 <= state(9);
                        yy8 <= state(8);
                        yy9 <= state(7);
                        yy10 <= state(6);
                        yy11 <= state(5);
                        yy12 <= state(4);
                        yy13 <= state(3);
                        yy14 <= state(2);
                        yy15 <= state(1);
        ELSE
                        yy0 <= FX;
                        yy1 <= FX;
                        yy2 <= FX;
                        yy3 <= FX;
                        yy4 <= FX;
                        yy5 <= FX;
                        yy6 <= FX;
                        yy7 <= FX;
                        yy8 <= FX;
                        yy9 <= FX;
                        yy10 <= FX;
                        yy11 <= FX;
                        yy12 <= FX;
                        yy13 <= FX;
                        yy14 <= FX;
                        yy15 <= FX;
        END IF;

        END PROCESS;
    y0 <= tstate_to_tri(f_state(yy0));
    y1 <= tstate_to_tri(f_state(yy1));
    y2 <= tstate_to_tri(f_state(yy2));
    y3 <= tstate_to_tri(f_state(yy3));
    y4 <= tstate_to_tri(f_state(yy4));
    y5 <= tstate_to_tri(f_state(yy5));
    y6 <= tstate_to_tri(f_state(yy6));
    y7 <= tstate_to_tri(f_state(yy7));
    y8 <= tstate_to_tri(f_state(yy8));
    y9 <= tstate_to_tri(f_state(yy9));
    y10 <= tstate_to_tri(f_state(yy10));
    y11 <= tstate_to_tri(f_state(yy11));
    y12 <= tstate_to_tri(f_state(yy12));
    y13 <= tstate_to_tri(f_state(yy13));
    y14 <= tstate_to_tri(f_state(yy14));
    y15 <= tstate_to_tri(f_state(yy15));
END full;
```

APPENDIX C - AM29510 BEHAVIOR MODEL

A. TIMING BEHAVIOR

```
-- AM29510.VHD
library SHU;
USE SHU.EPROM_PACK.ALL;
USE std.std_logic.ALL;
USE std.std_ttl.ALL;
USE work.intpack_ttl.ALL;
USE work.MINGPACK.ALL;
ENTITY AM29510 IS
    PORT    (x0,x1,x2,x3,x4,x5,x6,x7,
             x8,x9,x10,x11,x12,x13,x14,x15,
             y0,y1,y2,y3,y4,y5,y6,y7,
             y8,y9,y10,y11,y12,y13,y14,y15,
             tc,acc,sub_add,le_oe,
             clkx,clky,clkp : IN tri_state;
             p16,p17,p18,p19,p20,p21,p22,p23,
             p24,p25,p26,p27,p28,p29,p30,p31 : OUT tri_state);
END AM29510;

ARCHITECTURE full OF AM29510 IS

CONSTANT tma     :TIME:=50 ns;
CONSTANT ts      :TIME:=0 ns; --25
CONSTANT th      :TIME:=0  ns;--5
CONSTANT tsprel  :TIME:=25 ns;
CONSTANT thprel  :TIME:=0  ns;
CONSTANT tpwh    :TIME:=20 ns;
CONSTANT twpl    :TIME:=20 ns;
CONSTANT tpdp    :TIME:=40 ns;
CONSTANT tpdy    :TIME:=40 ns;
CONSTANT tphz    :TIME:=35 ns;
CONSTANT tplz    :TIME:=35 ns;
CONSTANT tpzh    :TIME:=40 ns;
CONSTANT tpzl    :TIME:=40 ns;
CONSTANT thcl    :TIME:=0  ns;

    COMPONENT multiplier_16_bit
        PORT  (x0,x1,x2,x3,x4,x5,x6,x7,
           x8,x9,x10,x11,x12,x13,x14,x15,
           y0,y1,y2,y3,y4,y5,y6,y7,
           y8,y9,y10,y11,y12,y13,y14,y15,
           tc,acc,sub_add,le_oe,
           clkx,clky,clkp : IN t_wlogic := F0;
```

```
        p16,p17,p18,p19,p20,p21,p22,p23,
        p24,p25,p26,p27,p28,p29,
        p30,p31 : OUT t_wlogic := F0);
    END COMPONENT;
    FOR ALL : multiplier_16_bit
        USE ENTITY work.multiplier_16_bit(full);

    signal  xx0,xx1,xx2,xx3,xx4,xx5,xx6,xx7,
        xx8,xx9,xx10,xx11,xx12,xx13,xx14,xx15,
        yy0,yy1,yy2,yy3,yy4,yy5,yy6,yy7,
        yy8,yy9,yy10,yy11,yy12,yy13,yy14,yy15,
        pp16,pp17,pp18,pp19,pp20,pp21,pp22,pp23,
        pp24,pp25,pp26,pp27,pp28,pp29,pp30,pp31,
        tc1,acc1,sub_add1,le_oe1,
        clkx1,clky1,clkp1 : t_wlogic := F0;

BEGIN

MUL : multiplier_16_bit
    PORT MAP  (xx0,xx1,xx2,xx3,xx4,xx5,xx6,xx7,
        xx8,xx9,xx10,xx11,xx12,xx13,xx14,xx15,
        yy0,yy1,yy2,yy3,yy4,yy5,yy6,yy7,
        yy8,yy9,yy10,yy11,yy12,yy13,yy14,yy15,
        tc1,acc1,sub_add1,le_oe1,clkx1,clky1,clkp1,
        pp16,pp17,pp18,pp19,pp20,pp21,pp22,pp23,
        pp24,pp25,pp26,pp27,pp28,pp29,pp30,pp31);

    -- input delay processing

    tc1    <= f_ttl(tri_to_tstate(tc));
    acc1   <= f_ttl(tri_to_tstate(acc));
    sub_add1 <= f_ttl(tri_to_tstate(sub_add));
    le_oe1 <= f_ttl(tri_to_tstate(le_oe));
    clkx1 <= f_ttl(tri_to_tstate(clkx));
    clky1 <= f_ttl(tri_to_tstate(clky));
    clkp1 <= f_ttl(tri_to_tstate(clkp));

    xx0 <= f_ttl(tri_to_tstate(x0));
    xx1 <= f_ttl(tri_to_tstate(x1));
    xx2 <= f_ttl(tri_to_tstate(x2));
    xx3 <= f_ttl(tri_to_tstate(x3));
    xx4 <= f_ttl(tri_to_tstate(x4));
    xx5 <= f_ttl(tri_to_tstate(x5));
    xx6 <= f_ttl(tri_to_tstate(x6));
    xx7 <= f_ttl(tri_to_tstate(x7));
    xx8 <= f_ttl(tri_to_tstate(x8));
    xx9 <= f_ttl(tri_to_tstate(x9));
    xx10 <= f_ttl(tri_to_tstate(x10));
    xx11 <= f_ttl(tri_to_tstate(x11));
    xx12 <= f_ttl(tri_to_tstate(x12));
    xx13 <= f_ttl(tri_to_tstate(x13));
```

```
    xx14 <= f_ttl(tri_to_tstate(x14));
    xx15 <= f_ttl(tri_to_tstate(x15));

    yy0 <= f_ttl(tri_to_tstate(y0));
    yy1 <= f_ttl(tri_to_tstate(y1));
    yy2 <= f_ttl(tri_to_tstate(y2));
    yy3 <= f_ttl(tri_to_tstate(y3));
    yy4 <= f_ttl(tri_to_tstate(y4));
    yy5 <= f_ttl(tri_to_tstate(y5));
    yy6 <= f_ttl(tri_to_tstate(y6));
    yy7 <= f_ttl(tri_to_tstate(y7));
    yy8 <= f_ttl(tri_to_tstate(y8));
    yy9 <= f_ttl(tri_to_tstate(y9));
    yy10 <= f_ttl(tri_to_tstate(y10));
    yy11 <= f_ttl(tri_to_tstate(y11));
    yy12 <= f_ttl(tri_to_tstate(y12));
    yy13 <= f_ttl(tri_to_tstate(y13));
    yy14 <= f_ttl(tri_to_tstate(y14));
    yy15 <= f_ttl(tri_to_tstate(y15));

    p16 <= tstate_to_tri(f_state(pp16)) AFTER tma+tpdp;
    p17 <= tstate_to_tri(f_state(pp17)) AFTER tma+tpdp;
    p18 <= tstate_to_tri(f_state(pp18)) AFTER tma+tpdp;
    p19 <= tstate_to_tri(f_state(pp19)) AFTER tma+tpdp;
    p20 <= tstate_to_tri(f_state(pp20)) AFTER tma+tpdp;
    p21 <= tstate_to_tri(f_state(pp21)) AFTER tma+tpdp;
    p22 <= tstate_to_tri(f_state(pp22)) AFTER tma+tpdp;
    p23 <= tstate_to_tri(f_state(pp23)) AFTER tma+tpdp;
    p24 <= tstate_to_tri(f_state(pp24)) AFTER tma+tpdp;
    p25 <= tstate_to_tri(f_state(pp25)) AFTER tma+tpdp;
    p26 <= tstate_to_tri(f_state(pp26)) AFTER tma+tpdp;
    p27 <= tstate_to_tri(f_state(pp27)) AFTER tma+tpdp;
    p28 <= tstate_to_tri(f_state(pp28)) AFTER tma+tpdp;
    p29 <= tstate_to_tri(f_state(pp29)) AFTER tma+tpdp;
    p30 <= tstate_to_tri(f_state(pp30)) AFTER tma+tpdp;
    p31 <= tstate_to_tri(f_state(pp31)) AFTER tma+tpdp;

END full;
```

B. MULTIPLICATION BEHAVIOR

```
-- MULTIPLY.VHD
USE std.std_logic.ALL;
USE std.std_ttl.ALL;
USE work.intpack_ttl.ALL;
ENTITY multiplier_16_bit IS
    PORT    (x0,x1,x2,x3,x4,x5,x6,x7,
             x8,x9,x10,x11,x12,x13,x14,x15,
             y0,y1,y2,y3,y4,y5,y6,y7,
```

```
            y8,y9,y10,y11,y12,y13,y14,y15,
            tc,acc,sub_add,le_oe,
            clkx,clky,clkp : IN t_wlogic;
            p16,p17,p18,p19,p20,p21,p22,p23,
            p24,p25,p26,p27,p28,p29,p30,p31 : OUT t_wlogic);
END multiplier_16_bit;

ARCHITECTURE full OF multiplier_16_bit IS

BEGIN
    -- operation

        PROCESS(clkx,clky,clkp)
        VARIABLE state : t_logarray(1 TO 16);
        VARIABLE x : integer := 0;
        VARIABLE y : integer := 0;
        VARIABLE p : integer := 0;
        VARIABLE accr,tcr,sub_add_r : t_wlogic;
        VARIABLE unknownx,unknowny : boolean := true;
        BEGIN

          -- pickup x data

      IF (f_rising_edge(clkx)) THEN
          state(16) := x0;
          state(15) := x1;
          state(14) := x2;
          state(13) := x3;
          state(12) := x4;
          state(11) := x5;
          state(10) := x6;
          state(9) := x7;
          state(8) := x8;
          state(7) := x9;
          state(6) := x10;
          state(5) := x11;
          state(4) := x12;
          state(3) := x13;
          state(2) := x14;
          state(1) := x15;
                IF tcr = F0 THEN
                    f_logictoint(state,unknownx,x);
                ELSE
                    f_logic_to_int2c(state,unknownx,x);
                END IF;
      END IF;

          -- pickup y data

         IF f_rising_edge(clky) THEN
                  state(16) := y0;
```

```
                    state(15) := y1;
                    state(14) := y2;
                    state(13) := y3;
                    state(12) := y4;
                    state(11) := y5;
                    state(10) := y6;
                    state(9)  := y7;
                    state(8)  := y8;
                    state(7)  := y9;
                    state(6)  := y10;
                    state(5)  := y11;
                    state(4)  := y12;
                    state(3)  := y13;
                    state(2)  := y14;
                    state(1)  := y15;
                              IF tcr = F0 THEN
                  f_logictoint(state,unknowny,y);
                              ELSE
                  f_logic_to_int2c(state,unknowny,y);
                              END IF;
               END IF;

        IF (f_rising_edge(clkx)
        OR f_rising_edge(clky)) THEN
IF ((NOT unknowny)AND(unknownx)
          AND(LE_OE=F1)AND(accr=F0)) THEN
    p := y;
          ELSIF (NOT unknownx) AND (NOT unknowny) THEN
                   IF (accr = F1) THEN
                      IF (sub_add_r = F1) THEN
                          p := x * y - p;
                      ELSE
                          p := p + x * y;
                      END IF;
                   ELSE       -- accr = F0
                      IF (LE_OE = F0) THEN
                          p := x * y;
                      ELSE
                          p := y;
                      END IF;
                   END IF;
              END IF;
              IF tcr = F0 THEN
                 f_inttologic(p,state,ttl);
              ELSE
                 f_int1_to_logic2c(p,state,ttl);
              END IF;
              accr := acc;
              tcr  := tc;
              sub_add_r := sub_add;
           END IF;
```

```
            -- assign values to output signals

             IF f_rising_edge(clkp) THEN
             IF (NOT unknowny) THEN
                  p16 <= state(16);
                  p17 <= state(15);
                  p18 <= state(14);
                  p19 <= state(13);
                  p20 <= state(12);
                  p21 <= state(11);
                  p22 <= state(10);
                  p23 <= state(9);
                  p24 <= state(8);
                  p25 <= state(7);
                  p26 <= state(6);
                  p27 <= state(5);
                  p28 <= state(4);
                  p29 <= state(3);
                  p30 <= state(2);
                  p31 <= state(1);
               ELSE
                  p16 <= FX;
                  p17 <= FX;
                  p18 <= FX;
                  p19 <= FX;
                  p20 <= FX;
                  p21 <= FX;
                  p22 <= FX;
                  p23 <= FX;
                  p24 <= FX;
                  p25 <= FX;
                  p26 <= FX;
                  p27 <= FX;
                  p28 <= FX;
                  p29 <= FX;
                  p30 <= FX;
                  p31 <= FX;
               END IF;
          END IF;
           END PROCESS;
END full;
```

APPENDIX D - COSTFUNCTION TESTBENCH

```
--COSTFUNCDFFCLKINV_CHIP3_CHIP1_CONTROL&ASSIGN_PIN_CHIP4.VHD
-- Costfunction test bench program
-- CHIP3 and CHIP1 are full 16/12 bit adder with register and
-- also has feedback to adder input.
-- They are the programmed EP1800.
-- CONTROL is an programmed EP1800 for state machine
-- and pc interface.
-- NOW connect CONTROL SIGNALS & PC_INTERFACE
-- NOW contain sizeprom and AM29510_MULTIPLIER/ACC
library EP1800LIB, SHU;
use EP1800LIB.EP1800_PACK.all, SHU.EPROM_PACK.all;
USE work.mingpack.ALL;
entity costfuncshu is
end costfuncshu;

architecture portion of costfuncshu is
    component EP1800
      generic (JEDEC : in string);
      port (pin_14,pin_15,pin_16,pin_17,
            pin_19,pin_20,pin_21,pin_22,
            pin_48,pin_49,pin_50,pin_51,
            pin_53,pin_54,pin_55,pin_56:in tri_state:='0';
            pin_2,pin_3,pin_4,pin_5,pin_6,pin_7,
            pin_8,pin_9,pin_10,pin_11,pin_12,pin_13,
            pin_23,pin_24,pin_25,pin_26,pin_27,pin_28,
            pin_29,pin_30,pin_31,pin_32,pin_33,pin_34,
            pin_36,pin_37,pin_38,pin_39,pin_40,pin_41,
            pin_42,pin_43,pin_44,pin_45,pin_46,pin_47,
            pin_57,pin_58,pin_59,pin_60,pin_61,pin_62,
            pin_63,pin_64,pin_65,pin_66,pin_67,pin_68
            :inout tri_state:='0');
    end component;
-- configuration specification
    for all : ep1800 use entity EP1800LIB.ep1800(structural);

    component AM29510
      port (x0,x1,x2,x3,x4,x5,x6,x7,
        x8,x9,x10,x11,x12,x13,x14,x15,
        y0,y1,y2,y3,y4,y5,y6,y7,
        y8,y9,y10,y11,y12,y13,y14,y15,
        tc,acc,sub_add,le_oe,
        clkx,clky,clkp : IN tri_state:='0';
        p16,p17,p18,p19,p20,p21,p22,p23,
        p24,p25,p26,p27,p28,p29,
        p30,p31 : OUT tri_state:='0');
```

```
    end component;
-- configuration specification
    for all : AM29510 use entity WORK.AM29510(FULL);

    component SIZEPROM
      port (x0,x1,x2,x3,x4,x5,x6,x7,
        x8,x9,x10,x11,x12,x13,x14,x15 : IN tri_state:='0';
        y0,y1,y2,y3,y4,y5,y6,y7,
        y8,y9,y10,y11,y12,y13,y14,y15 : OUT tri_state:='0');
    end component;
-- configuration specification
    for all : sizeprom use entity WORK.SIZEPROM(FULL);

    signal     INV1,CLR_REQ_L1,LATCH1 :tri_state;
    signal  CLK :tri_state := '0';
    signal  PCCLR_L1 :tri_state := '1';
    signal  X300_L,QOUT_L,QOUTE,QIN_L,X304_L,X306_L,
       X308_L,X30A_L,AEN,IOR_L,IOW_L :tri_state;
    signal  RAM_MAC,PIPE,CLKX,CLKY,CF_DONE :tri_state;
    signal  ENABLE,ENi_L,ENj_L :tri_state;
    signal  A: tri_vector(0 to 9) :=  "0000000001";
    signal  S: tri_vector(0 to 8) :=  "000000000";
    signal  T: tri_vector(0 to 15):=  "0000000000000000";
    signal Y: tri_vector(0 to 15):=  "0000000000000000";
    signal Q: tri_vector(0 to 9) :=  "0000000000";
    signal U: tri_vector(0 to 11):=  "000000000000";
    signal V: tri_vector(0 to 15):=  "0000000000000000";
    signal P: tri_vector(0 to 15):=  "0000000000000000";
    signal M: tri_vector(0 to 13):=  "00000000000000";
    signal INi: tri_vector(1 to 16):="0000000000000000";
    signal INj: tri_vector(1 to 16):="0000000000000000";
    signal Oi: tri_vector(0 to 15):= "0000000000000000";
    signal Oj: tri_vector(0 to 15):= "0000000000000000";
    signal  gnd: tri_state := '0';
    signal  vcc: tri_state := '1';
    signal eval : boolean := false;
begin

-- use named association interface list.
-- associated lists are according to
-- chip map from ALTERA Design Processor Utilization Report.

        CHIP3:EP1800 generic map("chip3.jed")
                port map
        (pin_2=>gnd,pin_3=>gnd,pin_4=>gnd,pin_5=>Y(1),
           pin_6=>Y(8),pin_7=>Y(12),pin_8=>Y(14),pin_9=>Y(15),
           pin_10=>T(1),pin_14=>CLR_REQ_L1,pin_15=>INV1,
           pin_16=>LATCH1,pin_17=>T(3),pin_19=>T(4),
           pin_20=>T(5),pin_21=>T(6),pin_22=>T(7),
           pin_23=>gnd,pin_24=>gnd,pin_25=>gnd,
           pin_27=>gnd,pin_28=>gnd,pin_29=>gnd,
```

```
        pin_30=>gnd,pin_31=>gnd,pin_32=>gnd,
        pin_33=>Y(9),
        pin_34=>Y(10),pin_36=>gnd,pin_37=>gnd,pin_38=>gnd,
        pin_39=>gnd,pin_40=>Y(2),pin_41=>Y(3),
        pin_42=>Y(4),pin_43=>Y(5),pin_44=>gnd,
        pin_45=>gnd,pin_46=>T(2),
        pin_48=>T(8),pin_49=>T(9),
        pin_50=>T(10),pin_51=>T(11),pin_53=>T(12),
        pin_54=>T(13),pin_55=>T(14),pin_56=>T(15),
        pin_61=>gnd,pin_62=>T(0),pin_63=>Y(0),
        pin_64=>Y(6),pin_65=>Y(7),
        pin_66=>Y(11),pin_67=>Y(13),pin_68=>gnd);

    CHIP1:EP1800 generic map("chip1.jed")
                port map
    (pin_2=>U(0),pin_3=>U(1),pin_4=>U(4),pin_5=>U(6),
        pin_6=>U(8),pin_7=>U(9),pin_8=>gnd,pin_9=>gnd,
        pin_14=>gnd,pin_15=>gnd,
        pin_16=>gnd,pin_17=>CLR_REQ_L1,pin_19=>INV1,
        pin_20=>LATCH1,pin_21=>Q(0),pin_22=>Q(1),
        pin_23=>gnd,pin_24=>gnd,pin_25=>gnd,pin_26=>gnd,
        pin_27=>gnd,pin_28=>gnd,pin_29=>gnd,
        pin_30=>gnd,pin_31=>gnd,pin_32=>gnd,
        pin_33=>gnd,
        pin_34=>gnd,pin_36=>gnd,pin_37=>gnd,pin_38=>gnd,
        pin_39=>gnd,pin_40=>gnd,pin_41=>gnd,
        pin_42=>gnd,pin_43=>gnd,pin_44=>gnd,
        pin_45=>gnd,pin_46=>gnd,pin_47=>gnd,
        pin_48=>Q(2),pin_49=>Q(3),
        pin_50=>Q(4),pin_51=>Q(5),pin_53=>Q(6),
        pin_54=>Q(7),pin_55=>Q(8),pin_56=>Q(9),
        pin_57=>gnd,pin_61=>gnd,
        pin_62=>U(2),pin_63=>U(3),pin_64=>U(5),
        pin_65=>U(7),
        pin_66=>U(10),pin_67=>U(11),pin_68=>gnd);

CONTROL:EP1800 generic map("control.jed")
            port map
    (pin_2=>gnd,pin_3=>gnd,pin_4=>gnd,pin_5=>gnd,
     pin_6=>gnd,pin_7=>gnd,pin_8=>gnd,pin_9=>gnd,
     pin_10=>S(0),pin_11=>S(1),pin_12=>S(2),
     pin_13=>S(3),
     pin_14=>gnd,pin_15=>A(0),
     pin_16=>A(1),
     pin_17=>A(2),pin_19=>A(3),
     pin_20=>A(4),pin_21=>A(5),pin_22=>A(6),
     pin_23=>S(8),pin_24=>X300_L,pin_25=>QOUT_L,
     pin_26=>QOUTE,
     pin_27=>gnd,pin_28=>gnd,pin_29=>gnd,
     pin_30=>gnd,pin_31=>gnd,pin_32=>gnd,
     pin_33=>gnd,
```

```
      pin_34=>QIN_L,pin_36=>gnd,pin_37=>gnd,pin_38=>gnd,
      pin_39=>gnd,pin_40=>gnd,pin_41=>gnd,
      pin_42=>gnd,pin_43=>gnd,pin_44=>X304_L,
      pin_45=>X306_L,pin_46=>X308_L,pin_47=>X30A_L,
      pin_48=>A(7),pin_49=>A(8),
      pin_50=>A(9),
      pin_51=>AEN,pin_53=>CLK,
      pin_54=>IOR_L,pin_55=>IOW_L,pin_56=>PCCLR_L1,
      pin_57=>S(4),pin_58=>S(5),pin_59=>S(6),
      pin_60=>S(7),pin_61=>gnd,
      pin_62=>gnd,pin_63=>gnd,pin_64=>gnd,
      pin_65=>gnd,
      pin_66=>gnd,pin_67=>gnd,pin_68=>gnd);


PROM_SIZE:SIZEPROM
      port map(U(0),U(1),U(2),U(3),U(4),U(5),U(6),U(7),
           U(8),U(9),U(10),U(11),gnd,gnd,gnd,gnd,
           V(0),V(1),V(2),V(3),V(4),V(5),V(6),V(7),
           V(8),V(9),V(10),V(11),V(12),
           V(13),V(14),V(15));

MULTIPLY:AM29510
      port map(V(0),V(1),V(2),V(3),V(4),V(5),V(6),V(7),
           V(8),V(9),V(10),V(11),V(12),
           V(13),V(14),V(15),
           Y(0),Y(1),Y(2),Y(3),Y(4),Y(5),Y(6),Y(7),
           Y(8),Y(9),Y(10),Y(11),Y(12),
           Y(13),Y(14),Y(15),
           RAM_MAC,RAM_MAC,vcc,vcc,CLKX,CLKY,CLKY,
           P(0),P(1),P(2),P(3),P(4),P(5),P(6),P(7),
           P(8),P(9),P(10),P(11),P(12),
           P(13),P(14),P(15));

   gnd <= '0';
   vcc <= '1';

   CLK <= bit_to_tri(NOT tri_to_bit(CLK)) AFTER 300 ns;
   PCCLR_L1 <= '1',
           '0' AFTER 10 ns,
           '1' AFTER 310 ns;

process(ENABLE,INV1,RAM_MAC)
begin
   if ENABLE='1' then
      if RAM_MAC='0' then
       T <= "0100000000000000"; -- ACC[i]
      else
       T <= "1111000000000000"; -- TABLE[i]
       Q <= "1111000000";       -- SIZE[i]
      end if;
```

```
            else
               if INV1='1' then
               T <= "1000000000000000"; -- TABLE[j]
               Q <= "1000000000";       -- SIZE[j]
               end if;
            end if;
        end process;

            CLR_REQ_L1 <= S(0);
            ENABLE     <= S(1);
            ENi_L      <= bit_to_tri(NOT tri_to_bit(ENABLE));
            ENj_L      <= ENABLE;
            LATCH1     <= S(2);
            INV1       <= S(3);
            RAM_MAC    <= S(4);
            CLKY       <= S(5);
            CLKX       <= S(6);
--          PIPE       <= S(7);
            CF_DONE    <= S(7);

            eval <= true AFTER 15 us;
--          ASSERT (S(7) /= '1')
            ASSERT (NOT eval)
               REPORT "SIMULATION DONE";

    end portion;
```

APPENDIX E - RESULT OF THE SIMULATION

AUG-22-1989 12:49:16 VHDL Report Generator PAGE 2

TIME (NS)	----SIGNAL NAMES---- STATE(0 TO 8)	CHIP3 T(0 TO 15)	CHIP1 Q(0 TO 9)	AM29510 P(0 TO 15)
0	"UUUUUUUUU"	"0000000000000000"	"0000000000"	"UUUUUUUUUUUUUUUU"
15	"000000000"			
335	"110000000"			
[illegible]		"0100000000000000"		
[illegible]	"110000001"			
1565	"111000100"			
2135	"010011000"			
[illegible]		"1111000000000000"	"1111000000"	
2255				"0100000000000000"
2705	"110010001"			
3335	"1[illegible]101000[illegible]"			
3785	"100110001"			
[illegible]		"10000[illegible]000000000"	"1000[illegible]"	
4565	"1[illegible]1110000"			
5135	"000011100"			
[illegible]				"0[illegible]00011000000000"
[illegible]	"[illegible]"			
[illegible]	"[illegible]"			
[illegible]				"[illegible]0001[illegible]11111111"
[illegible]	"[illegible]1000[illegible]00"			
[illegible]		"0100000000000000"		
[illegible]	"110000001"			
[illegible]	"[illegible]"			
[illegible]	"0100110[illegible]"			
[illegible]		"1111000000000000"	"1111[illegible]"	
[illegible]				"01000[illegible]0000000"
[illegible]	"[illegible]"			
[illegible]	"[illegible]"			
[illegible]	"[illegible]"			
[illegible]		"1000[illegible]0000000"	"1[illegible]"	
[illegible]	"1[illegible]111000"			
[illegible]	"[illegible]"			
[illegible]				"000[illegible]1100000000"

LIST OF REFERENCES

1. Wu, L.J., and Curtis, T.E., "Practical Graph Partitioning Algorithms for SONAR," *Proc ICASSP*, 1988.

2. Jensen, P.A., "Optimal Network Partitioning," *Operation Research*, v. 19, pp. 916-932, 1970.

3. *VHDL MANUAL*, 2nded., IEEE 1.76, 1989.

4. Lipsett, R., Schaefer, C.F., and Ussery, C., *VHDL: Hardware Description and Design*, Kluwer Academic Publishers, 1989.

5. ALTERA Corporation, *Application Handbook*, ALTERA, 1988.

6. Pollard, L.H., *Computer Design and Architecture*, Prentice Hall, 1990.

7. ALTERA Corporation, *User-Configurable Logic Data Book*, ALTERA, 1988.

8. SHU SHIH-MING, *EPLD model with VHDL*, Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1989.

9. Armstrong, J.R., *Chip Level Modeling with VHDL*, Prentice Hall, 1989.

10. Leung, S.S., and Shanblatt, M.A., *ASIC System Design with VHDL: A Paradigm*, Kluwer Academic Publishers, 1989.

11. Coelho, D.R., *The VHDL Handbook*, Kluwer Academic Publisher, 1989.

12. Barton, D.L., *A First Course in VHDL*, Intermetric Inc., 1989.